НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.75

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2019 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Спосіб знаходження обсягу вхідних даних для транспортування в системах розподілених волонтерських обчислень»

Виконав:
студент ІІ курсу, групи КП-81мп
Орос Богдан Богданович _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент
Онай Микола Володимирович _____

Керівник:
Ст. викладач кафедри ПЗКС, к.т.н.,
Рибачок Наталія Антонівна _____

Рецензент:
Доцент кафедри ІЕ ФЕЛ, к.т.н., доцент
Вунтесмері Юрій Володимирович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) – 121 «Інженерія програмного забезпечення» («Програмне забезпечення комп'ютерних та інформаційно-пошукових систем»)

ЗАТВЕРДЖУЮ

Науковий керівник кафедри

_____ І.А. Дичка

«___»_____2018 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Орос Богдану Богдановичу

1. Тема дисертації «Спосіб знаходження обсягу вхідних даних для транспортування в системах розподілених волонтерських обчислень», науковий керівник дисертації Рибачок Наталія Антонівна, к.т.н., старший викладач, затверджені наказом по університету від «13» листопада 2019 р. № 3895-с.
2. Термін подання студентом дисертації «12» грудня 2019 р.
3. Об'єкт дослідження: процес знаходження розміру проміжку вхідних даних для транспортування, ґрунтуючись на поведінкових особливостях волонтера.
4. Предмет дослідження: способи знаходження розміру вхідних даних для обчислень.
5. Перелік завдань, які потрібно розробити:
   - провести аналіз існуючих методів визначення обсягу даних для транспортування
   - дослідити способи отримання поведінкових особливостей користувача;
   - формалізувати отриманні поведінкові особливості користувачів у вигляді метрик;
   - застосувати вибрані метрики для виведення формул обчислення обсягу вхідних даних для транспортування;
   - застосувати отримані формули у програмному забезпеченні для проведення волонтерських розподілених обчислень.
6. Орієнтовний перелік публікацій
   - Тези доповіді "Спосіб знаходження обсягу вхідних даних для транспортування в системах розподілених волонтерських обчислень" XII наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2019)

7. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доц. каф. ПЗКС		

8. Дата видачі завдання «04» жовтня 2018 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Ґрунтовне ознайомлення з предметною галуззю	17.02.2019	
2.	Визначення структури магістерської дисертації; вивчення літератури, пошук додаткової літератури, патентний пошук	04.03.2019	
3.	Робота над першим розділом магістерської дисертації; проведення наукового дослідження	16.04.2019	
4.	Проведення наукового дослідження; робота над другим розділом магістерської дисертації; розроблення програмного забезпечення	14.05.2019	
5.	Проведення наукового дослідження; робота над статтею за результатами наукового дослідження	17.09.2019	
6.	Завершення роботи над основною частиною магістерської дисертації; підготовка ілюстративного матеріалу; підготовка матеріалів доповіді на конференції ПМК-2019	28.10.2019	
7.	Оформлення текстової і графічної частини магістерської дисертації	10.11.2019	

Студент _____ Б.Б. Орос

Науковий керівник дисертації _____ Н.А.Рибачок

РЕФЕРАТ

Актуальність теми. Волонтерські обчислення надають можливість науковим товариствам проводити велику кількість обчислень безкоштовно, що надає змогу підтверджувати гіпотези, проводити експерименти. Розподілені волонтерські обчислення за допомогою браузеру є доволі перспективним рішенням, враховуючи поширеність пристроїв з підключенням до мережі Інтернет. Проведення обчислень в браузері передбачає відсутність довіри до клієнтів, а також велику кількість комунікацій між серверною і клієнтською частинами. Для вирішення проблеми надлишкової комунікації пропонується вираховувати обсяг даних для транспортування ґрунтуючись на поведінкових особливостях користувача формалізованих у вигляді метрик. Зібрані метрики пропонується використовувати для отримання формул обчислення обсягу вхідних даних та подальшого їх використання в існуючій системі волонтерських розподілених обчислень. Такий спосіб надасть можливість повертати необхідний обсяг даних кожному користувачеві та зменшити навантаження на серверну частину у вигляді надлишкових запитів та блокувань.

Об’єкт дослідження – процес знаходження обсягу вхідних даних для транспортування ґрунтуючись на поведінкових особливостях волонтера.

Предмет дослідження – способи знаходження обсягу вхідних даних для подальшого транспортування.

Мета роботи – розробка способу знаходження обсягу проміжку вхідних даних для транспортування в системах розподілених волонтерських обчислень.

Методи дослідження – формалізація поведінкових особливостей користувача у вигляді метрик з подальшим їх використанням в формулах для отримання обсягу вхідних даних для транспортування.

Наукова новизна:

1. Запропоновано набір метрик, який доцільно використовувати для обчислення обсягу вхідних даних індивідуально під кожного користувача.
2. Розроблено спосіб знаходження обсягу вхідних даних для транспортування з використанням метрик, таким чином зменшуючи навантаження на серверну частину способом зменшення кількості мережевих запитів.

Практична цінність. Використання метрик для обчислення обсягу вхідних даних було інтегровано в існуючу систему для проведення розподілених волонтерських обчислень, що надало можливість зменшити кількість мережевих запитів вдвічі, а також реагувати на сповільнення та пришвидшення повернення результатів обчислювальною системою користувача-волонтера.

Апробація роботи. Результати досліджень доповідалися та обговорювалися на XIII науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2019, яка відбулась у листопаді 2019 р., та опубліковані у збірнику тез доповідей за результатами конференції.

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'ятьох розділів, висновків та додатків.

У вступі зроблено огляд предметної області дослідження, сформульовано множину проблем, що розглядаються та обґрунтовано актуальність напрямку досліджень.

У першому розділі проведено аналіз існуючих способів визначення обсягу вхідних даних для транспортування, визначено їх недоліки та переваги, а також описано принцип їх роботи. Зазначено, що метрики вже було запропоновано враховувати в аналогах у вигляді використання тривалості сесії обчислення. Зроблено висновок про критерії, які слід враховувати під час розроблення нового способу визначення обсягу даних на базі поведінкових особливостей користувачів.

У другому розділі аргументовано знаходження обсягу вхідних даних на основі користувацьких метрик, наведено способи отримання користувацьких особливостей та формалізовано їх у вигляді списку метрик. На основі запропонованих метрик, наведено приклади використання метрик у вигляді коефіцієнтів формули знаходження обсягу вхідних даних.

У третьому розділі описано процес інтеграції способу знаходження обсягу вхідних даних в існуючу систему для проведення волонтерських розподілених обчислень. Наведено приклади програмного коду, що застосовується для обчислення, а також проаналізовано отримані результати кількості мережевих запитів до та після інтегрування запропонованого способу.

У четвертому розділі наведено способи ручного та автоматизованого тестування розробленого програмного забезпечення, проаналізовано переваги та недоліки використання технологій тестування.

У п’ятому розділі проаналізовано можливості використання запропонованого способу не тільки в системах волонтерських обчислень, а й в поширених комерційних системах розподілених обчислень.

У висновках проаналізовано отримані результати роботи.

У додатках наведено копію презентації та лістинг фрагментів програмної реалізації.

Робота виконана на 68 сторінках, містить 16 рисунків, список використаних літературних джерел з 8 найменувань та 2 додатки.

Ключові слова: система розподілених волонтерських обчислень, метрики поведінкових особливостей користувача, обсяг вхідних даних для обчислення, кількість запитів до сервера, транспортування даних, web-worker, контейнеризація, Docker.

ABSTRACT

Actuality. Volunteer calculations enable scientific societies to carry out a large number of calculations free of charge, which allows them to confirm hypotheses and conduct experiments. Distributed volunteer computing with a browser is a very promising solution, given the prevalence of devices with an Internet connection. Browser computing involves a lack of trust in clients, as well as a great deal of communication between the server and client units. To solve the problem of overcommunication it is suggested to calculate the amount of data for transportation based on the behavioral characteristics of the user formalized in the form of metrics. It is proposed to use the collected metrics to obtain formulas for calculating the amount of input and then using them in the existing system of volunteer distributed computing. This method will allow to return the required amount of data to each user and reduce the load on the server part in the form of redundant requests and blocking.

Object of research is the process of finding the volume of input for transportation, based on the behavioral characteristics of the volunteer.

Subjects of research are the methods of finding the volume of input for later transportation.

Research objective is the development of a method for finding the volume of input for transportation in distributed volunteer computing systems.

Methods of research include formalizing behavioral features of the user in the form of metrics, followed by their use in formulas to obtain the volume of input for transportation.

Scientific novelty:

1. A set of metrics is proposed that can be used to calculate the amount of input individually for each user.
2. A method for finding the volume of input for transportation using metrics has been developed, thus reducing the load on the server side by reducing the number of network requests.

Practical value. The use of metrics to calculate the amount of input was integrated into the existing system for conducting distributed volunteer computing, which made it possible to reduce the number of network requests twice, as well as to respond to the slowdown and acceleration of the volunteer user computing system.

Approbation. The research results were reported and discussed at the 13th Scientific Conference of Undergraduate and Graduate Students in Applied Mathematics and Computing PMK-2019, held in November 2019, and published in the abstract of the conference.

Structure and contents of the thesis.

The master's thesis consists of an introduction, five chapters, conclusions and appendixes.

The introduction provides an overview of the subject area of the study, sets out many of the problems under consideration, and substantiates the relevance of the research direction.

The first section analyzes the existing methods for determining the amount of input for transportation, identifies their disadvantages and advantages, and describes the principle of their operation. It is noted that the use of metrics has already been proposed in analogues in the form of time use. It was concluded that

consideration should be given to developing a new method for determining volume based on user behavioral characteristics.

The second section justified finding the amount of input based on custom metrics, outlined ways to get custom features and formalized them as a list of metrics. There were also examples of using metrics as input coefficients.

The third section describes the process of integrating how to find the amount of input into an existing system for volunteer distributed computing. Examples of the code used to calculate were given, and the results obtained from the number of network requests before and after integrating the proposed method were analyzed.

The fourth section describes how to manually and automatically test software and analyzes the benefits and disadvantages of using test technologies.

In the fifth section, the possibilities of using the proposed method were analyzed not only in volunteer computing systems, but also in common commercial distributed computing systems.

The findings summarize the results obtained.

Appendices provide a copy of the presentation and listing of program implementation fragments.

The work is made on 68 pages, contains 16 drawings, a list of used literature sources with 8 references and 2 appendices.

Keywords: distributed volunteer exchange system, behavioral metrics users use, process survey input, number of entries to server, data transport, web worker, containerization, Docker.

СПИСОК ТЕРМІНІВ

Розподілені обчислення – програмний спосіб розв'язання довготривалих трудомістких обчислювальних завдань з використанням обчислювальної потужності систем (комп’ютерів), об'єднаних в мережу.

Волонтерські обчислення – тип розподілених обчислень, де користувач (власник комп’ютеру, котрий знаходиться в мережі) надає обчислювальні потужності власного комп’ютеру для виконання обчислень.

Користувач, волонтер – людина, що має на меті роботу з продуктом та прийняття участі в цифровому волонтерстві.

Фреймворк – програмна платформа, що являє собою каркас програмного забезпечення та набір інструментів, що надає змогу ефективно створювати програмні продукти.

Завдання для обчислення – містить набір вхідних даних, функцію для обчислення та певний набір налаштувань.

Активне завдання – завдання, яке знаходиться в стані обчислення в користувача.

Функція для обчислення – певна функція, що призначена для перетворення елементу набору вхідних даних у результат й використовуються для кожного елементу вхідного набору.

Набір вхідних даних – масив елементів, які потрібно трансформувати в певний результат (обчислити). Елементи розділені спеціальним символом, який використовується як службовий і не приймає участі в обчисленнях.

Проміжок набору вхідних даних – масив елементів набору вхідних даних, які користувач отримує за один запит. Кількість проміжків, що отримує користувач залежить від багатьох параметрів системи.

Ідентифікатор користувача – спеціальний ідентифікатор сесії користувача для визначення його серед інших користувачів системи. Система

використовує веб-сокет з'єднання як ідентифікатор по причині відсутності доменних ідентифікаторів.

Активна сесія користувача – час, коли користувач знаходиться на вкладці веб-ресурсу та займається обчисленнями певного завдання.

Графік навантаження на систему – графік навантаження на систему користувача, побудовані на основі швидкості роботи системи користувача. Дані переобчислюються кожні N секунд для більш точного відображення стану системи.

Швидкість системи користувача – емпірично отриманий набір даних про систему користувача, що містить можливу швидкість центрального процесору системи.

Транспортування даних – переміщення даних з однієї обчислювальної системи на іншу. В контексті даної магістерської дисертації під транспортуванням даних вважається переміщення вхідних даних завдання для обчислення за допомогою мережевих запитів з серверної на клієнтську частину.

Поведінкові особливості користувача – емпірично отримана статистична інформація про користувача, його обчислювальну систему. Є узагальненням поведінки користувача на веб-ресурсі під час проведення обчислення.

Метрики – формалізовані поведінкові особливості користувача для подальшого використання для виведення формул обчислення обсягу вхідних даних для транспортування.

Куки – сховище браузера для збереження секретної інформації користувача. Зазвичай використовується для збереження унікальних ідентифікаторів, токенів доступу тощо. Автоматично надсилаються браузером на серверну частину у випадку запиту на відповідний домен.

Сесія – ініціалізоване з’єднання між серверною та клієнтською частинами.

Web-Worker – технологія, що надає можливість виконувати програмний код в середовищі браузера, не блокуючи основний потік виконання.

ЗМІСТ

ВСТУП

Покращення в області проведення довготривалих обчислень може сприяти розвитку багатьох суміжних та залежних областей. Для пришвидшення обчислень та збільшення об’єму даних для обчислень використовується вертикальне та горизонтальне масштабування систем. Розподілені волонтерські обчислення за допомогою браузеру є доволі перспективним рішенням, враховуючи поширеність пристроїв з підключенням до мережі Інтернет.

Але, вирішуючи проблеми масштабованості систем із великими вхідними даними, розподілені волонтерські обчислення породжують низку нових питань, до яких входять проблеми безпеки та довіри, швидкодії та своєчасності отримання результатів.

Виконання розподілених обчислень на серверній частині практично нівелює питання швидкодії обчислень, адже відсутня основна проблема – транспортування даних між серверними машинами. Але для волонтерських обчислень дане питання гостро стоїть та перешкоджає широкому поширенню цієї технології.

Для вирішення даної проблеми необхідно знайти доцільний обсяг вхідних даних для транспортування для кожного волонтера. Для більш точного визначення необхідних об’ємів транспортування пропонується враховувати потужності обчислювальної машини волонтера, на яку транспортуються дані, та поведінкові особливості волонтера. Для подальшого аналізу та прийняття рішення доцільним буде також зберігання та використання статистичних даних про обчислення, які вже проведені волонтером.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ СПОСОБІВ ЗНАХОДЖЕННЯ ОБСЯГУ ВХІДНИХ ДАНИХ

1.1. Загальні відомості про волонтерські обчислення

Волонтерські обчислення – це тип розподілених обчислень, де виконання програмного коду проводиться на обчислювальних системах користувачів-волонтерів. Програмний код для виконання транспортується до користувачів по мережі та виконується на їх обчислювальних системах. Системи для проведення розподілених волонтерських обчислень – це клієнт-серверні застосунки (рис. 1.1).

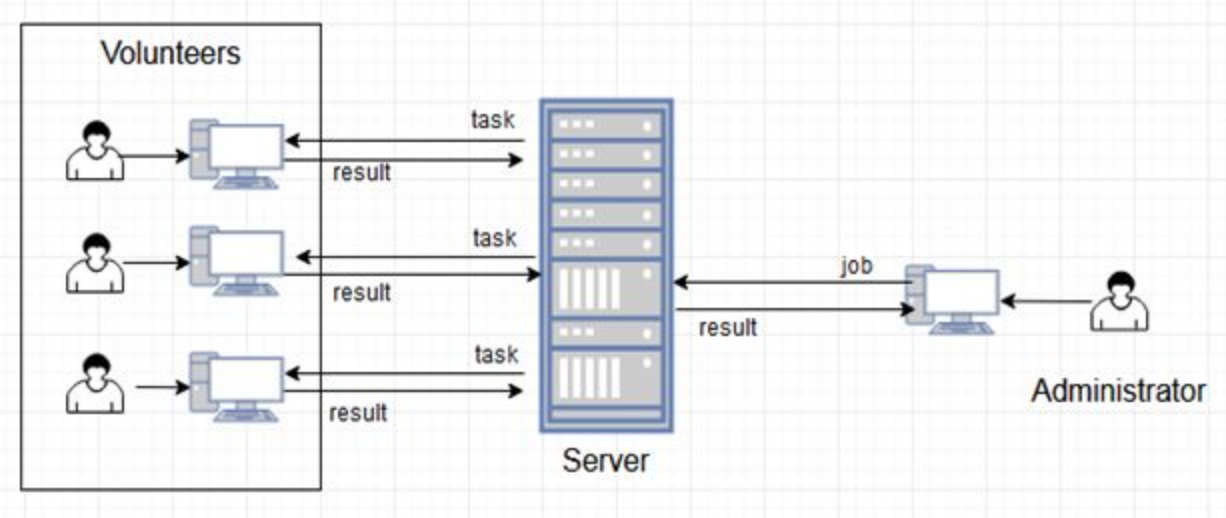


Рис. 1.1 Спрощена схема волонтерських обчислень

Проведення обчислень – зона відповідальності клієнтської частини. Це надає можливість масштабувати довготривалі обчислення не використовуючи додаткові обчислювальні потужності серверної частини. Роль сервера полягає в синхронізації обчислень між користувачами, зберіганні вхідних даних та результатів обчислень. Сервер повертає користувачам завдання для обчислення та вхідні дані.

Завдання для обчислення – це програмний код, що являє собою певну функцію мови програмування, яку необхідно виконати для отримання певних результатів, а вхідні дані для обчислення є аргументами цієї функції. В браузерних системах волонтерських розподілених обчислень, завдання для обчислення, вхідні дані та результати обчислень транспортуються по мережі. Користувачі-волонтери отримують завдання для обчислення при створенні активної сесії обчислення, тобто в момент приєднання до обчислення. Вхідні дані отримуються певними шматками.

У якості клієнтів наразі використовуються комп'ютерні додатки та веб-ресурси. У випадку використання комп'ютерних додатків, користувач встановлює його на власну обчислювальну систему, реєструється та приєднується до обчислення певного завдання. Застосунок створює з'єднання з серверною частиною, через яке буде відбуватись транспортування даних. Додаток також слідкує за станом з'єднання та перестворює його при необхідності. Завдання для обчислення отримується або під час оновлення програмного застосунку, або також транспортуються по мережі.

У випадку використання веб-платформи для проведення обчислення середовищем виконання програмного коду стає браузер.

Варто зазначити, що останні версії браузерів підтримують технологію Web Worker, що надає можливість виконувати програмний код в інших потоках, завдяки чому не блокується процес рендерингу сторінки користувача. Ця технологія також дозволяє проводити обчислення в декількох потоках.

Нижче наведено сценарій проведення розподілених волонтерських обчислень завдання в браузері з використанням Web Worker:

1. Адміністратор створює нове завдання: задає функцію, розміщує дані, налаштовує відповідну конфігурацію.

2. Система додає нове завдання (створює необхідні структури даних, ділить набір даних на проміжки, генерує сторінку із завданнями).
3. Волонтер робить запит до ресурсу, отримує згенеровану сторінку.
4. Волонтер обирає завдання.
5. Web Worker запитує проміжок для обчислень.
6. Система віддає проміжок для обчислень.
7. Web Worker виконує обчислення на проміжку.
8. Web Worker робить запит на збереження результатів обчислення на проміжку.
9. Система зберігає результат обчислення на проміжку.

Кроки 5-9 повторюються, поки є проміжки для обчислень по обраному завданню для цього волонтера, або волонтер не вирішить завершити обрахунок. Браузер знищує Web Worker наприкінці обчислень. Волонтер може продовжити обрахунок, тоді система надає йому нові проміжки для обчислення.

Якщо всі проміжки завдання обчислені, система позначає завдання, як завершене, адміністратор отримує можливість переглянути результат обчислень завдання.

Відповідна діаграма послідовності наведена на рис. 1.2.

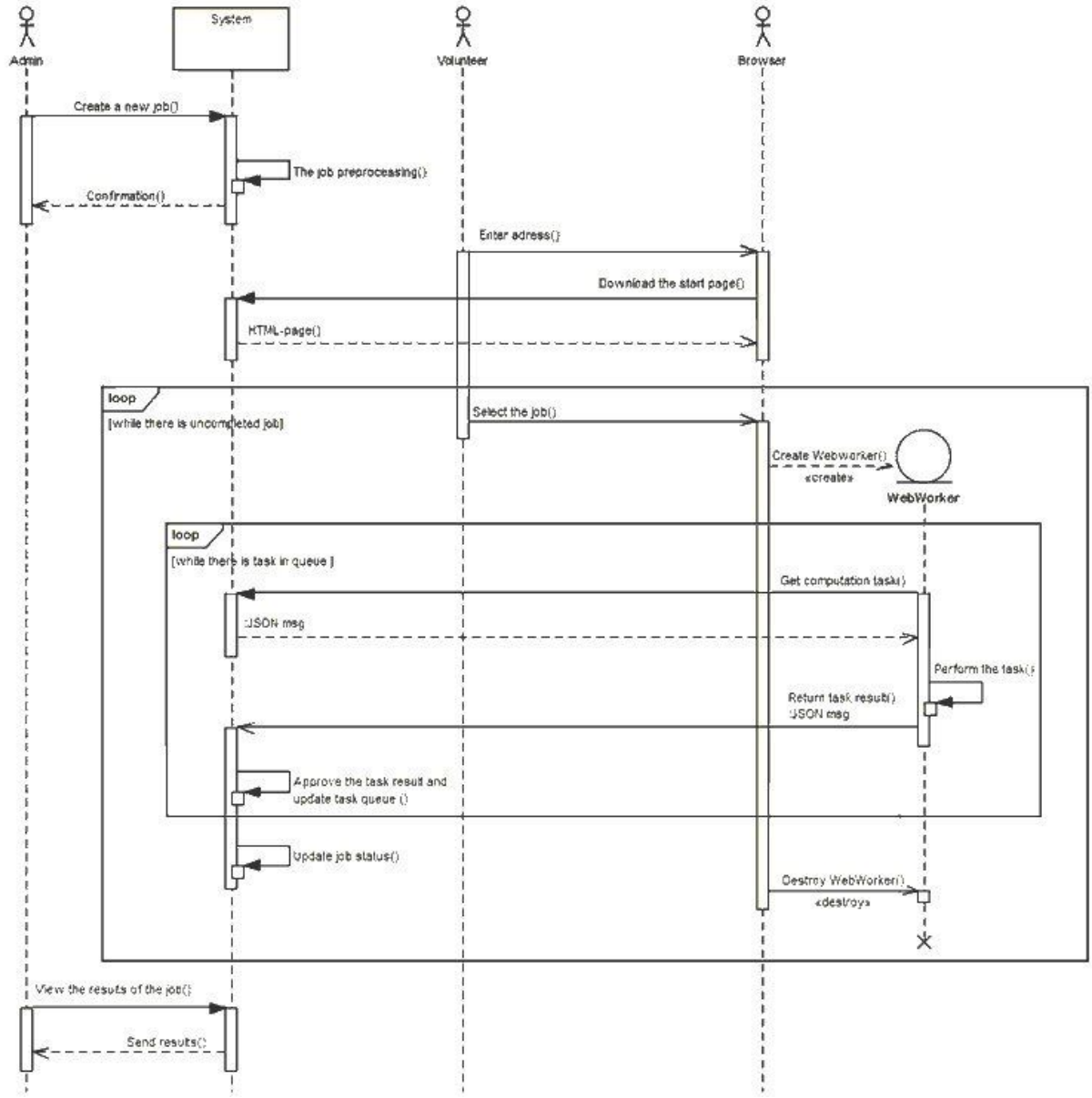


Рис. 1.2. Діаграма послідовності сценарію проведення волонтерських обчислень в браузері із Web Worker

1.2. Проблематика транспортування вхідних даних

Проведення волонтерських обчислень через браузер значно спрощує для кінцевих користувачів участь у волонтерських обчисленнях, але також накладає певні обмеження в транспортуванні даних на клієнтську частину. Відсутність довіри до користувача-волонтера потребує додаткового контролю над транспортуванням даних.

Незважаючи на різноманіття типів завдань для обчислення, варіантів виконання цих обчислень, формату та розмірів вхідних та вихідних даних – всі завдання для обчислення потребують певного уніфікованого алгоритму поділу на мінімальні самодостатні неподільні шматки. Дуже важливо, щоб дані шматки були поділені не тільки залежно від визначеного розробником розміру, а також мали логічні межі та не пересікались, тобто було прийнятними для подальшої обробки.

Найпростішим способом розбиття великого набору вхідних даних на шматки є використання вже існуючих методів розділення у поширених форматів зберігання та передачі даних: у випадку формату CSV – символ нового рядку, XML – використання елементів з однаковим ключем та еквівалентним рівнем вкладеності. Таке розбиття дасть можливість не тільки створити потрібні шматки на початковому етапі підготовки обчислень, а й надасть можливість розширювати набір вхідних даних вже під час проведення обчислень. Також ці шматки буде зручно перетворювати в необхідні для клієнтів формати у випадку використання різних технологій комунікації або форматів.

Також дуже важливою залишається можливість ідентифікації певного шматку в наборі вхідних даних. Тобто відразу відкидається ідея можливості використання шматків з довільною кількістю вхідних завдань для обчислень, адже система не зможе використовувати ефективні алгоритми для пошуку даних, а отже буде витрачати значно більший час на виконання запитів користувачів.

Спираючись на вимоги до визначення шматку вхідних даних, який надалі буде називатись проміжком вхідних даних, вже було визначено певні способи розбиття та подальшого транспортування набору вхідних даних.

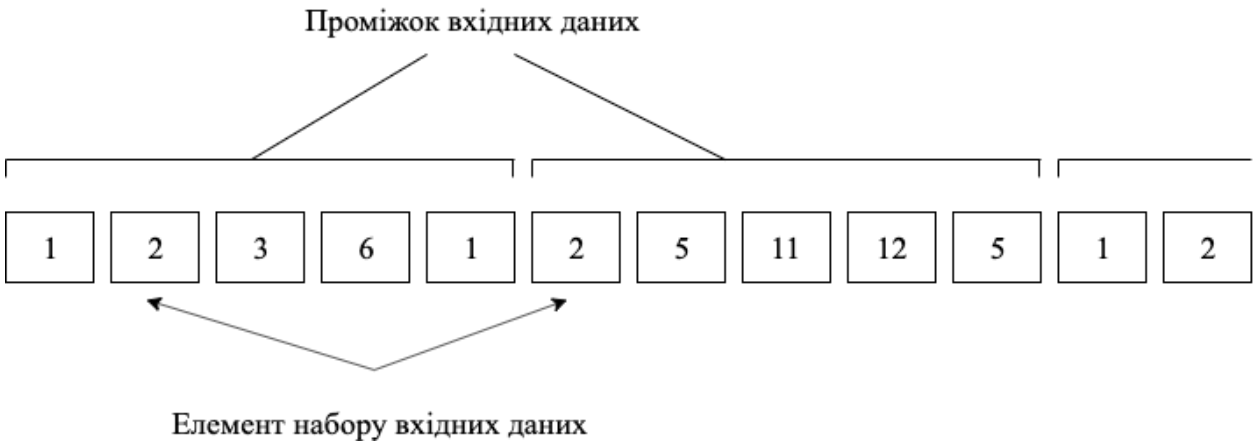


Рис 1.3. Влаштування проміжків на серверній частині

1.3. Використання проміжків одного розміру

Найпростішим способом розбиття множини вхідних даних обчислювального завдання є розбиття на проміжки однакового розміру.

Перед початком виконання обчислення множина вхідних даних розбивається на проміжки заданого розміру (рис. 1.4).

Користувачі-волонтери будуть отримувати проміжки однакового розміру незалежно від будь-яких обставин та характеристик.

Дане розбиття є дуже простим в реалізації та надає можливість провести велику кількість оптимізацій та підготовчих процесів.

Рис. 1.4. Транспортування проміжків одного розміру

Розбиття може бути явним і неявним. При явному розбитті множина дійсно розбивається на проміжки та зберігається вже в подрібненому вигляді. Даний спосіб потребує створення додаткових структур для зберігання інформації в сховищі даних й значно полегшує роботу з проміжками: з'являється можливість мати посилання на певні проміжки та створювати зв'язки між користувачами-волонтерами та проміжками; значно спрощується процес блокування певних проміжків завдяки використанню засобів мов програмування для створення блокувань. Недоліками явного розбиття є саме створення додаткових структур та підвищена складність лінивої ініціалізації проміжків, а отже процес поділу на проміжки може бути тривалим.

При неявному розбитті на проміжки, множина не розбивається на проміжки, присутня тільки абстракція проміжку як певної неподільної множини вхідних даних. В даному випадку користувачі-волонтери мають посилання тільки на певну позицію в множині, тобто робота з проміжками базується на зміщеннях. При використанні даного методу розбиття, система не витрачає час на додаткові підготовчі процеси, також відсутня необхідність в створенні додаткових структур. Система використовує значення зміщення для збереження результатів в сховище даних, а також для блокувань.

Зрозуміло, що використання проміжку одного розміру робить перерозрахунок проміжків майже неможливим без зупинення процесу обчислення.

Ефективність даного методу цілком залежить від правильно підібраного розміру проміжку. Використання даного методу не передбачає використання даних про кількість активних користувачів, адже підготовчі процеси відбуваються до отримання необхідної інформації. При використанні проміжків надто великого розміру, система буде блокувати проміжки вже при невеликій кількості волонтерів, що негативно вплине на час проведення

обчислення. Також в даному випадку користувач-волонтер повинен змінити завдання для обчислення, адже часто знаходитиметься в заблокованому стані та не буде проводити обчислення. Використання занадто малих проміжків також створить проблему надлишкових блокувань та великих витрат на транспортування даних з серверної частини на клієнтську.

Також даний метод передбачає, що складність обчислення одного проміжку є практично ідентична іншим проміжкам. Це не є проблемою при відсутності кворуму як способу перевірки коректності результатів обчислення, але при його наявності деякі проміжки можуть занадто довго залишатися в стані обчислення й бути заблокованими.

Але при вдало підібраному розміру проміжків реалізація перевірки коректності результатів через метод кворуму є дуже простою, адже перевірці підлягають проміжки ідентичного розміру й відсутня проблема перетину множин.

Найбільшим недоліком даного методу є відсутність гнучкості під час проведення обчислення. Ефективність проведення обчислення цілком залежить від вибору розміру проміжку вхідних даних. Без використання статистичних даних або певного набору підказок від адміністратора, система не може підлаштуватись під певне обчислювальне завдання та буде використовувати налаштування за замовчуванням. Використання певного набору підказок від адміністратора потребує глибокого аналізу завдання та витрат часу, а отже й вносить значний людський фактор в проведення обчислення, тому використання даного покращення краще уникнути.

1.4. Використання тривалості сесії для збільшення проміжків

Найпростішим способом адаптуватися під користувача-волонтера є використання певних метрик. Найпростішою метрикою за замовчуванням є тривалість сесії обчислення.

Її зміст полягає в тому, що при ініціюванні користувачем обчислення, серверна частина зберігає час початку сесії обчислення та оновлює його кожен раз при отриманні нового проміжку користувачем. Тривалість сесії проведення обчислення слугує коефіцієнтом для розрахунку кількості проміжків, що отримує користувач-волонтер для обчислення (рис. 1.5).

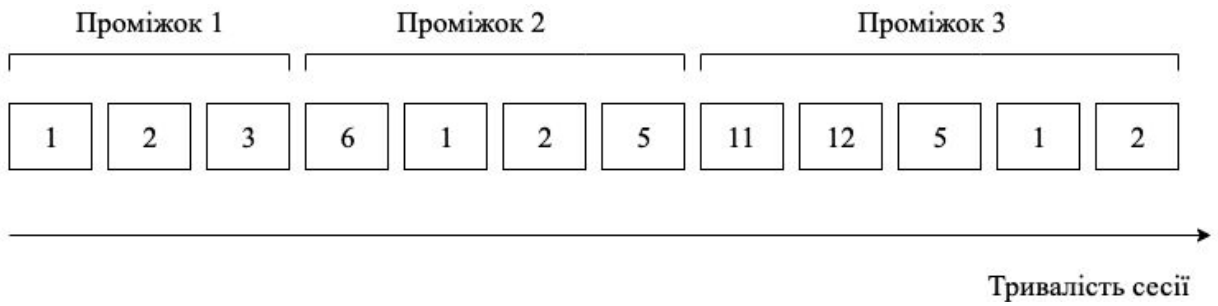


Рис. 1.5. Транспортування проміжків в залежності
від тривалості обчислення

Даний метод є вдосконаленням для існуючого методу використання проміжків однакового розміру. Але, на відміну від останнього, метод з коефіцієнтом потребує проміжків значно меншого розміру для ефективного використання часового коефіцієнту.

Перевагою даного вдосконалення є простота реалізації та хороший приріст швидкодії для довготривалих сесій користувачів-волонтерів з потужними системами.

Середня тривалість сесії може зберігатися на серверній частині для повторного використання. Але в такому випадку серверна частина також повинна вміти розпізнати користувача, що не завжди є можливою опцією для систем розподілених волонтерських обчислень.

Також даний метод має верхнє допустиме значення кількості отриманих проміжків для довготривалих сесій, щоб уникнути блокування великої кількості проміжків одним користувачем. Дане обмеження також є дуже доцільним при перевірці коректності результатів.

Недоліками даного методу є складність обробки виключних ситуацій при роботі з користувачами-волонтерами та односторонній погляд на тривалість сесії.

Під складністю обробки виключних ситуацій мається на увазі недетермінізм в рішеннях щодо покинутих сесій. В ситуації занадто довгого очікування результатів для певних проміжків від користувача системі необхідно вирішувати, чи є дана сесія обчислення покинутою. Отже, при довготривалих сесіях обчислень випадкові затримки результатів від користувачів можуть призвести до значних втрат результатів та надлишкових блокувань проміжків. Для вирішення даної проблеми використовуються такі методи:

- Відпускання блокування певним користувачем для уможливлення повторного обчислення певних проміжків іншим користувачем-волонтером. У випадку використання перевірки коректності обчислення дане покращення умісне, але при відсутності проблеми може виникати велика кількість переобчислень. При отриманні результатів (навіть із запізненням) сесія продовжується й збережена тривалість сесії зберігається системою для подальшого використання.

- Використання періодичних запитів до системи користувача для перевірки його стану. При отримання будь-якої помилки вважається, що користувач припинив обчислення, а отже його сесія вважається закінченою й необхідно встановити тривалість сесії за замовчуванням.

Однобокість погляду на тривалість сесії полягає в розгляді тривалості обчислення тільки як ефективності. У випадку слабкої системи користувача-волонтера тривалість сесії буде збільшуватись навіть для невеликої кількості опрацьованих проміжків та отриманих результатів. Для вирішення даної проблеми запропоновано використовувати кількість обчислених проміжків як коефіцієнт (метрику) замість використання часу часової тривалості активної сесії. Використання кількості обчислених проміжків виглядає значно доцільнішим, адже надає певну статистику про потужності користувача-волонтера у випадку можливості його ідентифікації.

1.5. Ручне введення необхідних параметрів

Даний метод використовується в системах розподілених обчислень з ідентифікованими довіреними користувачами. Оперуючи необхідними даними про користувачів, система задає відповідні значення про довжини проміжків для кожного окремого користувача. Даний метод є простим в реалізації, адже не використовуються будь-які обчислення та адаптації кількості проміжків, не відбувається збереження статистичних даних про сесії користувачів. Також даний метод надає найвищу пропускну здатність.

Під ручним введенням необхідних параметрів мається на увазі:

- Використання статичних розмірів проміжків для певного завдання прямо в програмному коді.
- Введення необхідних параметрів під час створення завдання.
- Використання файлів налаштувань з необхідними параметрами.

Недоліком даного методу є необхідність повного контролю над кластером розподілених систем. Також з'являється необхідність створення системи блокування користувача у випадку появи проблем під час обчислень.

Використання даного способу є також важливим у випадку обов'язкової ідентифікації користувача-волонтера (наприклад, наявність системи аутентифікації з даними про систему користувача). Проте в існуючих системах даний спосіб не використовується по причині відсутності можливості перевірки коректності введених користувачем даних.

1.6. Висновки

Розглянуті методи можуть використовуватися в системах розподілених волонтерських обчислень для вирішення проблеми з транспортуванням даних на клієнтську частину, але гостро відчувається необхідність у використанні додаткових метрик, які надали б можливість підлаштовуватися до особливостей сесії користувача.

Використання методу ручного введення необхідних параметрів є сумнівним для створення системи волонтерських обчислень в браузері через відсутність довіри та необхідність ефективної та обов'язкової ідентифікації користувача.

Використання методу проміжків одного розміру є компромісним варіантом для вирішення проблеми транспортування, але повністю нівелює навіть наявні метрики серверної частини. Тому в сучасних системах розподілених волонтерських обчислень використовується покращений метод з коефіцієнтом тривалості обчислення. Проте поза увагою залишається велика кількість метрик, які можливо зібрати під час проведення обчислення, а отже й адаптувати процес проведення обчислень під користувачів.

Необхідним є знаходження достатнього набору метрик, що нададуть можливість покращити транспортування даних без внесення додаткових ускладнень в процес та впливу на швидкодію обчислень.

РОЗДІЛ 2. ФОРМУЛЮВАННЯ МОДИФІКАЦІЇ СПОСОБУ ОБЧИСЛЕННЯ ОБСЯГУ ВХІДНИХ ДАНИХ ДЛЯ ТРАНСПОРТУВАННЯ

2.1. Аргументація вибору способу

2.1.1. Отримання поведінкових особливостей користувачів

Для подальшої формалізації поведінкових особливостей користувача необхідно визначити яким чином дані особливості будуть отримані та збережені. Програмний інтерфейс браузера надає дуже обмежені можливості для визначення теоретичної швидкодії системи користувача, що робить оцінку можливостей дуже приблизною. Наразі в найпопулярніших браузерах, що використовують рушій Chromium (до числа яких входить Google Chrome), можна отримати тільки кількість ядер центрального процесора системи користувача, що унеможливлює точну оцінку швидкодії системи.

Для отримання метрик серверною частиною, клієнтська частина колекціонує метрики та надсилає їх на серверну частину тільки при створенні початкової сесії для проведення обчислення певного завдання та при надсиланні результатів обчислення у вигляді метаданих. Таким чином не відбувається додаткових запитів на серверну частину тільки заради збору метрик.

2.1.2. Поведінкові особливості користувача отримані за сесію обчислення

Найпростішим шляхом отримання певних статистичних даних про користувача є аналіз його сесії обчислення. Під сесію обчислення вважається процес обчислення завдання з початку створення з’єднання з серверною частиною по проведенню обчислень та закриттям даного з’єднання через

будь-які причини (закриття вкладки браузера, перебої в зв’язку). Під час існування сесії серверна частина може ідентифікувати користувача стандартними засобами бібліотек роботи зі з’єднаннями.

Під час активної сесії обчислення серверна частина може зберігати кількість проміжків, які вже обчислив користувач. Клієнтська частина може порахувати час, що витрачається на обчислення одного проміжку та повернути отримані дані про час разом з результатами обчислення на серверну частину. Серверна або клієнтська частини можуть порахувати середній час проведення обчислення для певного проміжку або медіану, відкинувши таким чином дуже різкі відхилення для певних проміжків, під час обчислення яких система користувача була перевантажена тощо.

2.1.3 Використання збережених даних про сесії користувача

Дуже важливим є вміння розпізнати постійного користувача системи, адже це надасть можливість використовувати вже попередньо отримані особливості користувача, а отже й обчислити необхідний обсяг даних для транспортування значно швидше. Таким чином постає питання можливості ідентифікації користувача для подальшого збереження та перевикористання його поведінкових особливостей.

Реалізація ідентифікації користувача повністю залежить від системи для проведення волонтерських обчислень. При використанні веб-ресурсу можливо використовувати сторінки авторизації або зробити ідентифікацію незалежною від користувача.

При використанні сторінки авторизації з’являється значно більше можливостей:

- ідентифікація користувача в різних браузерах та на різних обчислювальних системах;

- ідентифікатор користувача постійний;
- ідентифікатор зберігається на серверній частині і не буде втрачений після чистки користувачем браузерного кешу.

Якщо створення додаткових кроків перед проведенням обчислення є недоцільним, то для ідентифікації користувача можна використати можливості браузера, а саме cookies. Необхідно згенерувати унікальний ключ та зберегти його в браузері. При повторній сесії обчислення, клієнтська система повинна надсилати збережений ідентифікатор серверній частині у вигляді метаданих. Даний ідентифікатор буде присутній тільки в одному браузері, в якому він був згенерований, а також буде видалений при очищенні cookies веб-ресурсу для проведення розподілених волонтерських обчислень.

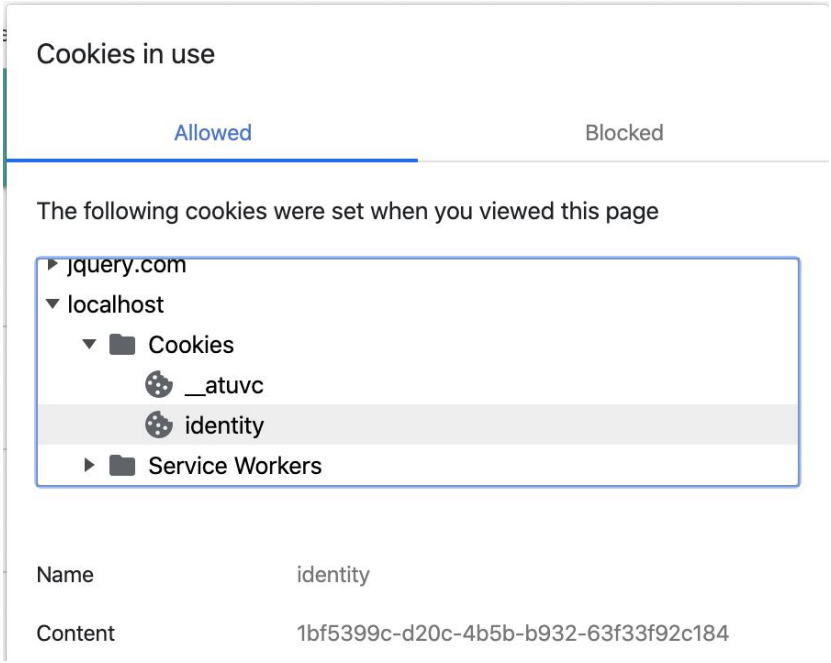


Рис. 2.1. Використання згенерованого ідентифікатора користувача

Серверна частина може зберігати отримані поведінкові особливості користувача та звертатись до них під час різних сесій. Важливо зазначити, що в системах волонтерських обчислень, що використовують систему

ідентифікації користувача тимчасовим унікальним ключем (за допомогою cookies), варто певним чином видаляти збережені поведінкові особливості користувачів, ідентифікатори яких вже є втраченими.

2.1.4. Параметри конфігурації користувача

Параметри конфігурації користувача – налаштування, що виставляються користувачем для контролю над проведенням обчислення. До них входять навантаження на систему користувача тощо.

При використанні браузера для проведення довготривалих обчислень, потрібно зауважити про особливості реалізації виконання програмного коду. Для написання програмного коду, що буде використовуватись в браузері, використовується мова програмування JavaScript. Інтерпретація команд (виконання програмного коду) відбувається в одному потоці, тому довготривале виконання коду може перешкоджати волонтеру користуватись власною системою.

Більшість сучасних браузерів надають можливість використовувати веб-воркери – засіб виконання програмного коду в окремому потоці. Таким чином можна проводити обчислення в незалежному потоці, а результати обчислень надавати основному потоку сторінки за допомогою повідомлень. Також варто зазначити, що використання веб-воркерів надає можливість проводити обчислення декількох завдань одночасно, створивши по веб-воркеру для кожного завдання, до якого приєднався користувач. Таку поведінку варто контролювати, адже постійне збільшення кількості потоків виконання має негативний вплив на швидкість виконання обчислень.

Значно важчою є ситуація у випадку відсутності веб-воркерів. Необхідно виконувати обчислення у потоці, який буде напряму працювати з користувачем.

Для виконання будь-якої довготривалої дії, браузер використовує техніку виконання програмного коду, що описується словом асинхронність. Довготривалі дії виконуються браузером з певними перервами на обробку інших подій, що могли виникнути - наприклад, натискання користувачем кнопки. Таким чином потік може реагувати на дії користувача і не заважатиме користуванню комп’ютером.

Можна скористатись тим самим механізмом та обчислювати кожний вхідний набір даних функції у вигляді асинхронного завдання, а між завданнями встановлювати штучну затримку.

При використанні таких затримок, серверна частина повинна врахувати тривалість цих затримок для визначення обсягу вхідних даних. Чим вища затримка між обчисленнями, тим менша кількість проміжків повинно повертатись на клієнтську частину.

Варто зауважити, що при формуванні метрик, які будуть використовуватись системою волонтерських обчислень, налаштування користувача є одними з найважливіших та повинні мати найвищий пріоритет.

2.1.5. Інформація від браузера про апаратні можливості системи користувача

Хоча браузер і не надає багато інформації про апаратні можливості системи користувача, все таки є можливість дізнатись про швидкодію системи користувача, використовуючи додаткові засоби.

Даний спосіб було розроблено сучасними компаніями для визначення можливостей системи користувача й поліпшення його досвіду користування сервісом.

Для визначення системи, де встановлений браузер, стандартом визначається заголовок “User-Agent”, що надсилається в кожному HTTP

запиті на серверну частину. В значенні даного заголовку вказується система, з якої надсилається запит, браузер, що використовується, а у випадку використанні мобільного телефону – ідентифікатор його моделі.

```
GET / HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.
0.3904.108 Safari/537.36
```

Рис. 2.2. Запит із заголовком “User-Agent”

За допомогою даного ідентифікатора, маючи таблицю результатів тестів швидкодії кожного мобільного пристрою, можна дуже точно визначити його швидкодію. Зазвичай, такі таблиці містять інформацію про швидкодію виконання коду на одному ядрі процессора, а також результати виконання коду усіма ядрами.

Знаючи обмеження браузера у виконанні програмного коду на сторінці одним потоком, необхідно використати результати тестів для одного ядра процессора та створити таблицю співставлення значення швидкодії системи користувача до кількості проміжків, що будуть транспортуватись на клієнтську частину.

Використання даного способу є доцільним для мобільних браузерів, адже значення їх заголовку не містить ідентифікатор комп’ютерної системи.

2.1.6. Способи збереження метрик

Важливо зазначити також про способи збереження метрик серверною частиною. Реалізація способу збереження метрик повністю залежить від того,

як дані метрики будуть використовуватись для перерахунку обсягу вхідних даних.

При використанні тільки сесійних поведінкових особливостей користувачів немає необхідності використовувати надійні та складні бази даних. Метрики можна зберігати в структурах даних в пам’яті серверу, що обробляє запити. Основним критерієм є швидкість доступу по ключу до необхідної інформації для зменшення впливу на швидкодію виконання коду серверною частиною через використання метрик.

При використанні метрик ідентифікованих користувачів доцільність використання надійного сховища залежить від важливості отриманих метрик.

2.2. Формалізація особливостей користувача у вигляді метрик

2.2.1. Визначення списку метрик для використання

Отримані поведінкові особливості користувач необхідно формалізувати у вигляді метрик для подальшого їх використання в системах волонтерських обчислень.

Пропонується для використання такий набір метрик:

- Кількість обчислених проміжків за сесію.
- Кількість обчислених проміжків в середньому за всі існуючі сесії користувача.
- Швидкість обчислення проміжку за сесію.
- Середня швидкість обчислення проміжку за всі існуючі сесії користувача.
- Відношення правильних результатів обчислення до помилкових.
- Встановлений користувачем рівень навантаження на систему.
- Інформація від браузера про апаратні можливості користувача.

Було формалізовано метрики, які можна отримати в середовищі

браузера та інтегрувати в існуючу систему для проведення розподілених волонтерських обчислень.

2.2.2. Рекомендації із застосування метрик

Очевидно, що не всі метрики можуть принести користь і не всі метрики є доцільними для використання в будь-якій системі. Тому важливим є використання обмеженої кількості метрик, що найкраще підходять для виконання поставленої задачі.

Найкращим способом буде використання різного набору метрик для різних користувачів, якщо таке можливо. Користувачі мобільних пристроїв можуть бути легко визначені та проаналізовані системою, в результаті чого будуть отримувати найоптимальнішу кількість проміжків відразу під час утворення сесії обчислення.

У випадку, якщо ідентифікація користувача є неможливою, варто використовувати метрики, що стосуються активної сесії обчислення. Такі метрики є найлегшими як для отримання, так і для опрацювання.

При використання користувацьких налаштувань варто надати їм найбільше уваги й встановити найвищий пріоритет, адже виставлені користувачем значення визначених параметрів можуть бути цілком протилежними емпірично отриманим метрикам. До визначених параметрів можна віднести навіть кількість проміжків, що будуть транспортуватись на клієнтську частину, але дана опція потребує достатньої довіри до системи користувача-волонтера, що є практично неможливим при використанні веб-ресурсів для проведення розподілених волонтерських обчислень.

Якщо система підтримує перевірку коректності результатів обчислення, для обчислення обсягу транспортування також варто застосувати відношення правильних результатів обчислення до помилкових. Зрозуміло, що

користувачі, які надсилають некоректні результати повинні бути опрацьовані виключним шляхом, але на початку пропонується значно обмежити кількість проміжків, що буде отримувати такий користувач.

Також необхідно проаналізувати метрики, що використовують агрегацію даних. В залежності від способу отримання результатів серверною частиною (отримання результатів для всього проміжку чи для кожного вхідного набору функції для обчислення), буде доцільним використовувати середнє значення метрики за певну кількість проміжків або медіану. Середнє значення доцільно використовувати, якщо результати передаються на серверну частину один раз для всіх обчислених проміжків, адже тоді не важлива різниця часу обчислення кожного проміжку (проміжки все рівно залишаються заблокованими). Використання медіани доцільне для визначення аномально довготривалого обчислення певних проміжків та зменшення впливу таких випадків на обчислення кількості проміжків.

2.3. Застосування метрик для виведення формул обчислення обсягу

При проведенні волонтерських обчислень замість надання сталого значення кількості проміжків серверна частина буде надавати кількість проміжків визначених за формулою, коефіцієнти в якій пропонується визначати на основі описаних вище метрик.

2.3.1. Використання кількості обчислених проміжків та середній час обчислення

Було проведено моделювання роботи системи волонтерських обчислень, в якій кількість даних для транспортування волонтерам визначається на основі двох метрик: кількості обчислених проміжків за сесію та середньої швидкості обчислення проміжку за сесію.

Для обчислення кількості неподільних проміжків r запропоновано формулу:

$$r = minAmount * (1 + \varepsilon_n) * a_n, \quad (2.1)$$

де n – кількість проміжків обчислених волонтером за сесію,

$minAmount$ – мінімальна кількість проміжків, що надається волонтеру сервером,

ε_n – коефіцієнт для визначення продуктивності системи волонтера.

Коефіцієнт ε_n визначено на основі обраних метрик:

$$\varepsilon_n = n / \sum_{i=0}^{n} t_i, \quad (2.2)$$

де t_i – середня тривалість обчислення проміжку.

Для обчислення даного коефіцієнту також можна використати медіану, для отримання результатів без врахування аномально довгих та коротких сесій обчислення.

Для відслідковування змін коефіцієнту ε_n, який вказує на пришвидшення чи зниження продуктивності обчислень, використовується відношення його теперішнього та попереднього значень.

$$a_n = \varepsilon_n / \varepsilon_{n-1}. \quad (2.3)$$

2.3.2. Використання збережених даних

Доцільно буде використовувати певні статистичні дані для отримання формул. Використання статистичних даних не впливатиме на кінцеву кількість проміжків, які будуть транспортуватись користувачу, але надасть можливість швидше дійти до кількості проміжків, що була обчислена за формулою (2.1).

Для користувачів, яких вдалось ідентифікувати, замість значення *minAmount* варто використати медіану кількості проміжків, які вони опрацьовують за сесію:

$$r = M(N) * (1 + \varepsilon_n) * a_n, \qquad (2.4)$$

де N – кількість обчислених проміжків за кожною з сесій обчислення.

Таким чином система буде ефективніше повертати проміжки відразу на початку проведення обчислень певним користувачем.

Варто зазначити, що для використання даної статистики необхідно також враховувати ідентифікатор обчислювальної системи, адже ідентифікований користувач може використовувати декілька систем. Нехтування даним ідентифікатором може призвести до транспортування більшої кількості проміжків, які здатна обчислити система користувача, що відрізняється потужністю від попередньої, за яку була отримана статистика.

Також постає питання використання медіани та середнього значення.Використання медіани надає можливість відкинути занадто довготривалі та короткочасні сесії обчислення, але є доцільним тільки у випадку великої кількості попередніх сесій обчислення користувача. Якщо на серверній частині відсутня необхідна кількість статистичних даних про користувача, то варто використовувати середня значення для обчислення початкової кількості проміжків.

2.3.3. Використання табличних даних

У випадку ідентифікації апаратних можливостей користувача, а саме унікального стандартизованого ідентифікатора його системи, замість використання значення *minAmount,* як початкової кількості проміжків, можна використати результат функції $g(i)$, що повертає попередньо визначену кількість проміжків для користувача. Підбір доцільної функції залежить від

системи для проведення розподілених обчислень та її користувачів. Тоді формула для обчислення кількості неподільних проміжків r матиме такий вигляд:

$$r = g(i) * (1 + \varepsilon_n) * a_n, \quad (2.5)$$

де i – унікальний стандартизований ідентифікатор системи.

Прикладом використання функції $g(i)$ може бути функція отримання потужності обчислювальної системи користувача. У випадку використання стаціонарних комп'ютерів, отримання інформації про систему користувача є обмеженим, тому пропонується використовувати кількість ядер центрального процесору систему як аргумент функції.

У випадку використання мобільних пристроїв пропонується використовувати числові значення потужності, які отримуються шляхом проведення тестів потужності центрального процесора. Ці числові характеристики є публічною інформацією, тому пропонується використовувати їх у функції для отримання початкової кількості проміжків. Найменше числове значення потужності може відповідати *minAmount*, наступні значення можна визначити пропорційно.

2.3.4. Використання рівня навантаження на систему користувача

У випадку використання метрик, які отримані на основі налаштувань користувача, наприклад навантаження на систему, кількість проміжків для транспортування буде визначатися за формулою:

$$R = r * k, \quad (2.6)$$

де r – кількість проміжків, отриманих в результаті обчислення основною формулою (2.1) для визначення кількості проміжків,

k – встановлений користувачем рівень навантаження на систему в проміжку (0.0, 1.0].

Використання вказаної формули надасть можливість підлаштувати кількість отриманих проміжків під кожного користувача. Переобчислення кількості проміжків може відбуватись при кожному запиті на отримання вхідних даних для обчислення, або коли серверна частина буде вважати це переобчислення доцільним. Очевидно, що перерозрахунок кількість проміжків на кожен запит не тільки значно точніше визначатиме необхідну кількість, але й створюватиме додаткове навантаження на серверну частину.

2.3.5. Використання метрик серверної частини

Також варто розглянути ситуацію, коли для обчислення кількості проміжків доцільно використати метрики, якими оперує тільки серверна частина. До таких метрик можливо віднести кількість одночасних користувачів-волонтерів, що займаються обчисленням певного завдання, або підозрілу активність користувача.

Як підозрілу активність користувача можна розглядати велику кількість наданих ним неправильних результатів, або дуже швидкий час отримання результатів обчислення. Такий випадок може спостерігатись, якщо користувач робитиме запити до серверної частини або в обхід браузера, надсилаючи запити на серверну частину вручну, або змінить програмний код клієнтської частини.

Варто обмежити максимальну кількість проміжків, які може отримати користувач за один запит. Результуюча формула отримання кількості проміжків буде мати вигляд:

$$R = r \wedge maxAmount, \tag{2.7}$$

де r – кількість проміжків, отриманих в результаті обчислення основною формулою (2.1) для визначення кількості проміжків,

maxAmount – максимальна кількість проміжків, які можуть бути транспортовані користувачу за один запит.

2.4. Висновки

Визначені формалізовані поведінкові особливості користувачів-волонтерів у вигляді метрик було використано у формулах для обчислення кількості проміжків, що будуть повертатись користувачу.

Також було визначено певні обмеження для серверної частини для підвищення безпеки транспортування занадто великої кількості проміжків.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАПРОПОНОВАНОГО СПОСОБУ В СИСТЕМІ РОЗПОДІЛЕНИХ ВОЛОНТЕРСЬКИХ ОБЧИСЛЕНЬ

3.1. Аналіз існуючого програмного забезпечення

Для перевірки доцільності використання отриманих формул, було вирішено використати їх для обчислення кількості проміжків в існуючій системі для проведення розподілених волонтерських обчислень.

Було проаналізовано програмну реалізацію системи, розробленою мною в якості дипломного проекту освітньо-кваліфікаційного рівня “Бакалавр”. Вона має клієнт-серверну архітектуру, серверна частина написано мовою програмування Java. В якості фреймворку було обрано SparkJava, що надало можливість зробити простий застосунок, сфокусувавшись тільки на доменній частині.

Вимогами для виконання коду серверної частини є середовище для виконання Java застосунків. Варто зазначити, що також можливо використовувати контейнеризацію, уникаючи необхідність налаштування середовища зі встановленою Java.

Було проаналізовано організацію завдань для обчислення на серверній частині для кращого розуміння можливостей оптимізації розбиття на проміжки. Перед виконанням завдання проходить етап підготовки, де проходить початкове розбиття на мінімальні проміжки. Це надає можливість повертати вже підготовлені проміжки під час створення активних сесій обчислення перших користувачів, що приєднались до новоствореного завдання. Архітектура побудована таким чином, що це не вплине на інтеграцію нових модулів обчислення кількості проміжків.

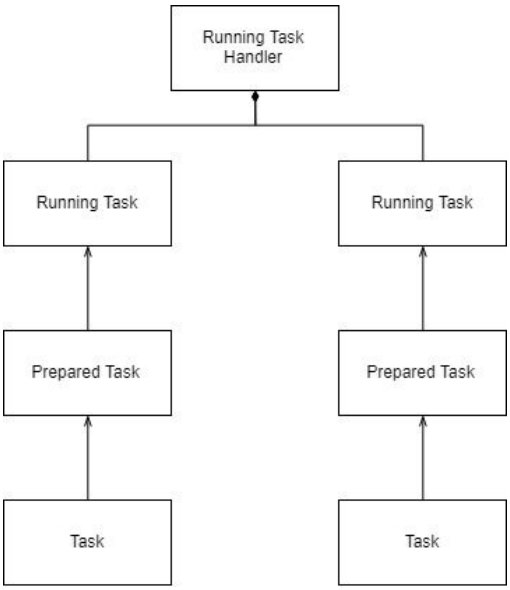


Рис. 3.1. Організація виконуваних завдань

Також було опрацьовано архітектуру існуючого застосунку, проаналізовано межі модулів. Застосунок побудований за моделлю MVC і присутні різні слої, що мають власні зони відповідальності. Варто зазначити, що представлення даних в застосунку та сховищі (базі даних) відрізняються, тому було використано технологію ORM для спрощення роботи з реляційними базами даних.

Рендерингом веб сторінок займається вбудована в фреймворк бібліотека, що значно спрощує роботу зі сторінками веб-ресурсу. Бібліотека надає можливість створювати веб сторінки власною мовою шаблонування, передавати в шаблонізатор дані у вигляді словнику, таким чином надаючи можливість підлаштовувати сторінки під індивідуальних користувачів.

Бібліотеки та програмний код клієнтської частини також встановлюються за допомогою посилань на JavaScript файли.

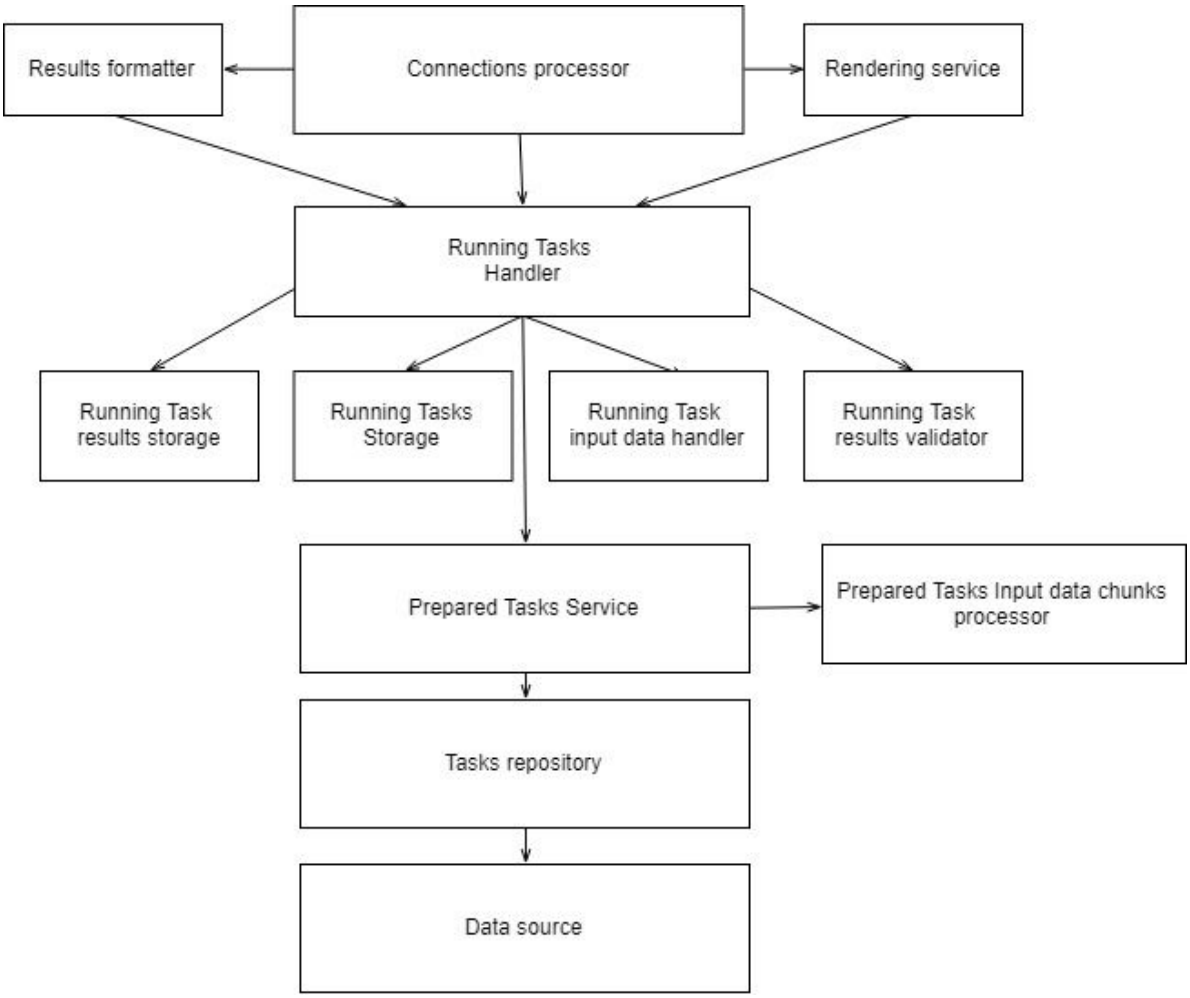


Рис. 3.2. Архітектура існуючого програмного продукту

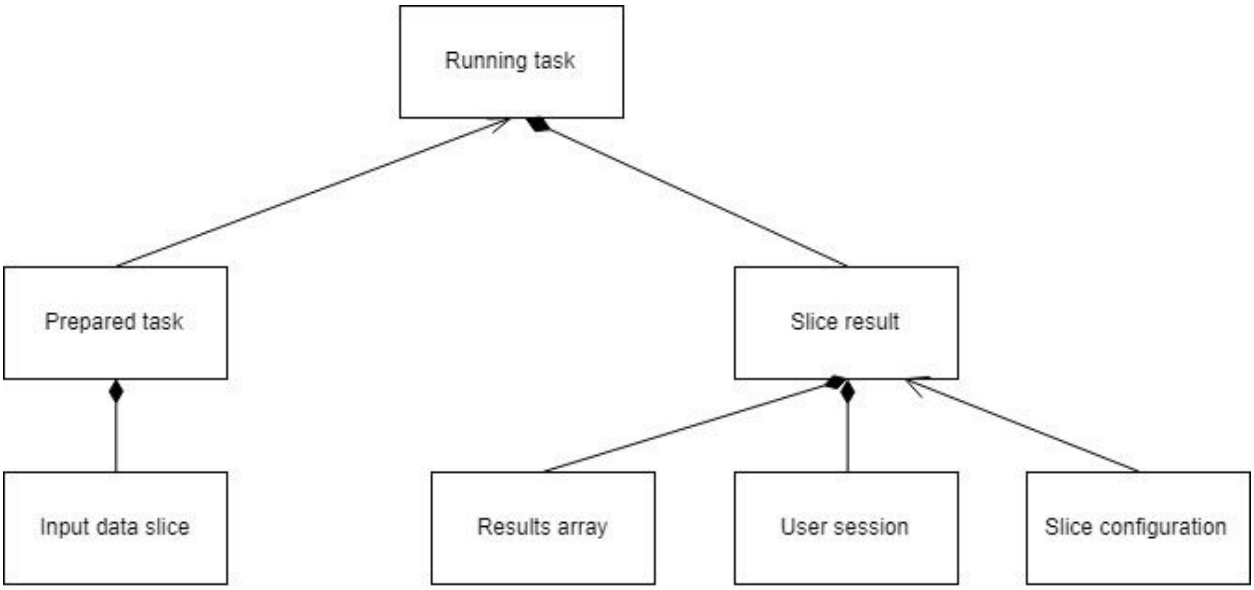


Рис. 3.3. Структура виконуваного завдання

Спершу було знайдено всі програмні інтерфейси, що потребують внесення змін. Зміни програмного коду відбулись як на клієнтській, так і на серверній частинах. Прийнято рішення зберегти вже використаний підхід в розробці програмного забезпечення у вигляді модульної архітектури.

Модульна архітектура надає можливість змінювати тільки певні компоненти не змінюючи програмний код та поведінку інших модулів. Її особливостями є організація взаємодії між модулями – на основі сталих програмних інтерфейсів, зміни яких мають відбуватись тільки у вигляді створення нових методів, а не в зміні існуючих. Також важливою особливістю є можливість підміни модулю на інший модуль, що реалізує ідентичний контракт. До переваг даної архітектури можна також віднести зручність програмного тестування, тобто написання автоматичних тестів мовою програмування для перевірки коректності виконання. Також варто зазначити, що модуль не повинен залежати від реалізації та внутрішньої структури інших модулів.

Для визначення кількості проміжків за допомогою формул необхідно було розроблено декілька модулів:

- модуль роботи з метриками на клієнтській частині;
- модуль обробки та збереження метрик на серверній частині;
- модуль розрахунку кількості проміжків за допомогою зібраних метрик.

Описані модулі замінили існуючі модулі в системі.

3.1.1. Модуль роботи з метриками на клієнтській частині

Існуючий модуль клієнтської частини системи для проведення розподілених волонтерських обчислень комунікує з серверною частиною за допомогою HTTP запитів. При створенні сесії для проведення обчислення, серверна частина повертає початковий проміжок для обчислення.

Обчислення відбувається за допомогою технології веб-воркерів. Після проведення обчислення, клієнтська частина надсилає результати в тілі HTTP запиту й отримує відповідь з новим проміжком для обчислення.

Було необхідно додати модуль для збору метрик під час сесії обчислення, засіб ідентифікації користувача, а також додати певні налаштування у вигляді рівня навантаження на систему користувача.

Для збору метрик було використано шаблон проектування “Декоратор”, який надає можливість дати додаткові функціональні можливості вже існуючим програмних методам зберігши їх програмний інтерфейс методу та не модифікувавши вже написаний програмний код.

Використання шаблонів проектування надало можливість значно зменшити зусилля для проведення інтеграції у вже існуючий програмний код, а також надало можливість проводити безпечні зміни навіть при відсутніх тестових сценаріях.

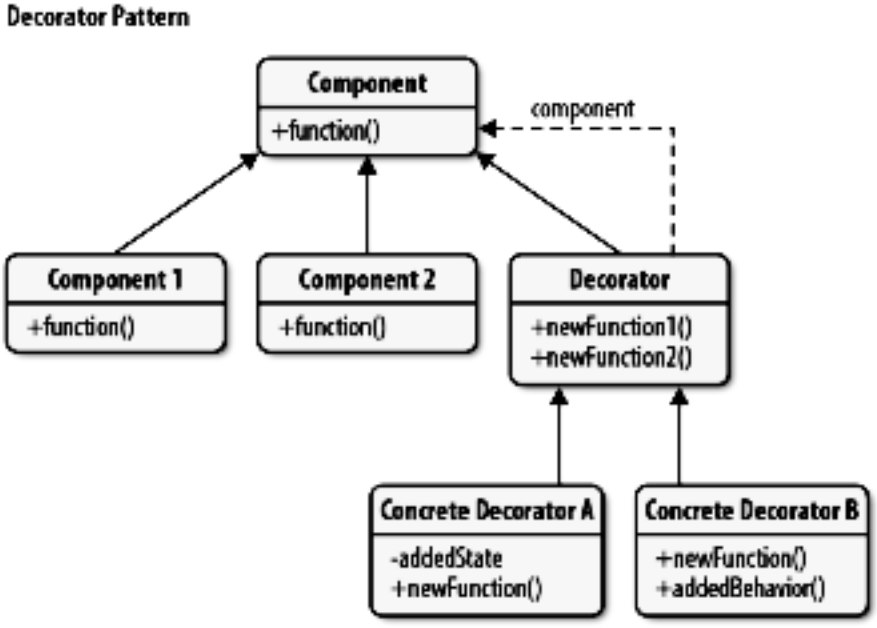


Рис. 3.4. Шаблон проектування декоратор

```
function processActiveTaskFunction(task) {
    const result = {};
    let counter = 0;
    let size = input.length;
    return async function () {
        while (counter < size) {
            if (counter < size) {
                let str = input[counter];
                result[str.data] = func(str.data);
                counter++;
                await sleep((Math.floor( x: Math.random() * slowRatio) + slowRatio ) * 10);
            }
        }
        return {
            action: 'result',
            result: {result: result, task: task.id}
        };
    };
}
```

Рис. 3.5. Код проведення обчислення без збору метрик

```
function processActiveTaskFunction(task) {
    const result = {};
    let counter = 0;
    let size = input.length;
    return async function () {
        while (counter < size) {
            if (counter < size) {
                let str = input[counter];
                result[str.data] = func(str.data);
                counter++;
                await sleep((Math.floor( x: Math.random() * slowRatio) + slowRatio ) * 10);
            }
        }
        return {
            action: 'result',
            result: {result: result, task: task.id}
        };
    };
}

function processActiveTaskFunctionDecoratedWithPerf(task) {
    const before = performance.now();
    const result = processActiveTaskFunction(task).call();
    const after = performance.now();
    result.metadata = {
        time: (after - before) / 1000,
        identity: getIdentity(),
    };
    return result;
}
```

Рис. 3.6. Код проведення обчислення зі збором метрик

Перед та після проведення обчислення певного набору вхідних даних відбувається замір часу методами виміру швидкодії мови програмування JavaScript.

Метаінформація відправляється на серверну частина разом з результатами в тілі HTTP запиту.

```
calculatedSlicesPerUser.put(request.identity,
        calculatedSlicesPerUser.getOrDefault(request.identity, defaultValue: 0L) + resultPayload.result.result.size());
calculationTimeInSeconds.put(request.identity,
        calculationTimeInSeconds.getOrDefault(request.identity, defaultValue: 0.0) + resultPayload.time);
```

Рис. 3.7. Код збереження метрик серверною частиною

Для ідентифікації користувача було вирішено не створювати додаткових сторінок авторизації та реєстрації. Найпростішим рішенням було генерування унікального ідентифікатору UUID у вигляді атрибуту cookies. Дані атрибути відправляються на серверну частину автоматично засобами браузера у випадку якщо адреса сторінки відправника співпадає з адресою серверної частини, на яку робиться запит.

```
function uuidv4() {
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {
    var r = Math.random() * 16 | 0, v = c == 'x' ? r : (r & 0x3 | 0x8);
    return v.toString(16);
  });
}
```

Рис. 3.8. Код генерування унікального ідентифікатора

Але для збереження існуючого контракту було вирішено додатково додати даний ідентифікатор в блок метаінформації тіла HTTP запиту до серверної частини засобами мови JavaScript.

Зміну навантаження на систему користувача можна регулювати за допомогою простої форми на сторінці виконання завдання. Коефіцієнт навантаження також передається на серверну частині в блоці метаінформації.

3.1.2. Модуль обробки та збереження метрик на серверній частині

Серверна частина також потребувала внесення змін в обробку запиту користувача. Результати обчислення отримуються в тілі HTTP запиту у вигляді JSON повідомлення. Це дало можливість додати обробку додаткового об’єкту метаінформації без внесення значних змін в обробку результатів обчислення.


```
{
  "content": {
    "tasks": [
      {
        "id": "inputSlice1",
        "result": "100.0"
      },
      {
        "id": "inputSlice2",
        "result": "114.3"
      }
    ]
  }
}
```


Рис. 3.9. Тіло запиту з результатами до внесення змін


```
"content": {
  "tasks": [
    {
      "id": "inputSlice1",
      "result": "100.0"
    },
    {
      "id": "inputSlice2",
      "result": "114.3"
    }
  ]
},
"metadata": {
  "identity": "3ec0ac93-78d3-44bc-a6cd-31bd3a72fc1f",
  "time": 12312.31
}
```


Рис. 3.10. Тіло запиту з результатами після внесення змін

Збереження метрик відбувається в новому створеному модулі. Було вирішено не використовувати зовнішні сховища для спрощення розробки та зменшення впливу на швидкодію існуючого рішення. Метрики зберігаються в словнику, де ключем є ідентифікатор користувача, а значенням – отримані метрики. Після завершення сесії обчислення метрики користувача потрапляють в інший словник зі статистичними метриками попередніх сесій обчислення.

3.1.3. Модуль розрахунку кількості проміжків за допомогою зібраних метрик

В останню чергу було розроблено модуль для розрахунку кількості проміжків за допомогою зібраних метрик. Було розроблено декілька програмних функцій, кожна з яких реалізує певну описану формулу. Функція обирається в залежності від користувача – це надає можливість використовувати різні формули та ще більше підлаштовуватись під поведінкові особливості користувачів-волонтерів.

```
SlicesCountCalculator slicesCountCalculator =
        new SlicesCountCalculator(calculatedSlicesPerSession, timeSpentPerSession);

if (isUserIdentified) {
    slicesCountCalculator.invoke(userStats);
} else {
    slicesCountCalculator.invoke(minAmount);
}
double slicesCount = slicesCountCalculator.count;
```

Рис. 3.11. Код визначення кількості проміжків

3.2. Отримані результати

Для отримання результатів проведено моделювання виконання обчислень в системі до та після внесення змін. Під час моделювання

вирішено встановити певне значення штучної затримки замість виконання обчислення для більш точного відслідковування кількості проміжків, що повертаються на серверну частину. Значення штучної затримки можна змінювати під час проведення обчислення за допомогою засобів для розробника в браузерах.

Для спостереження за метриками серверної частини (кількість запитів, кількість надісланих метрик) було використано програмний інструмент Prometheus. Він надає можливість візуалізувати отримані метрики у вигляді графіків. Особливістю цього програмного інструменту є спосіб отримання метрик. На відміну від звичайного підходу, коли серверна частина надсилає метрики самостійно, Prometheus отримує метрики через мережеві запити. Серверна частина надає програмний інтерфейс у потрібному форматі. Кожен рядок містить назву метрики та числове значення.

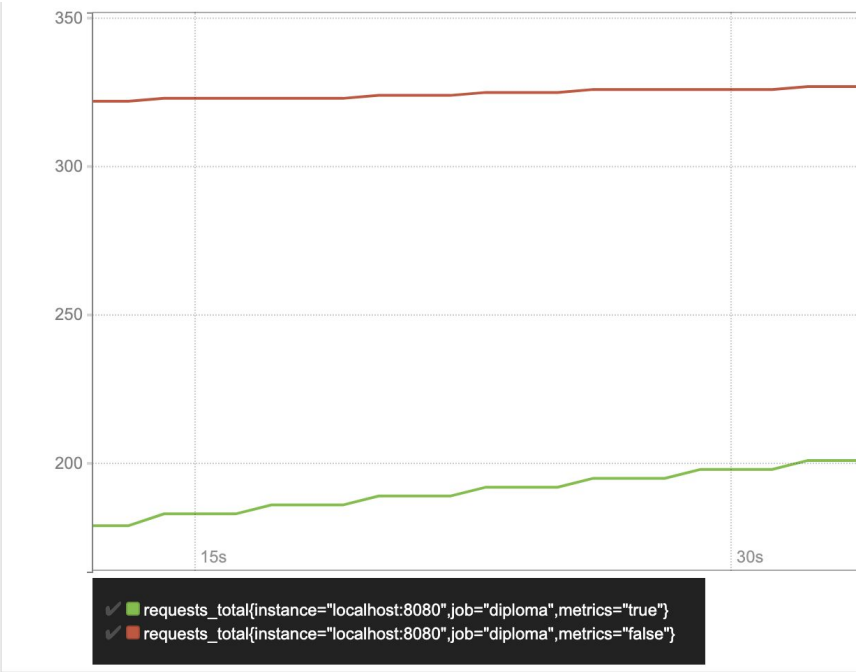


Рис. 3.12. Кількість запитів на серверну частину

Під час моделювання було проведено обчислення із використанням проаналізованих та запропонованого методів. Вхідні дані для обчислень містили 250 проміжків та встановлене значення *minAmount* = 10 на серверній частині. Дане моделювання ігнорує встановлене користувачем значення навантаження на систему. Статистичні дані користувача також не використовувались.

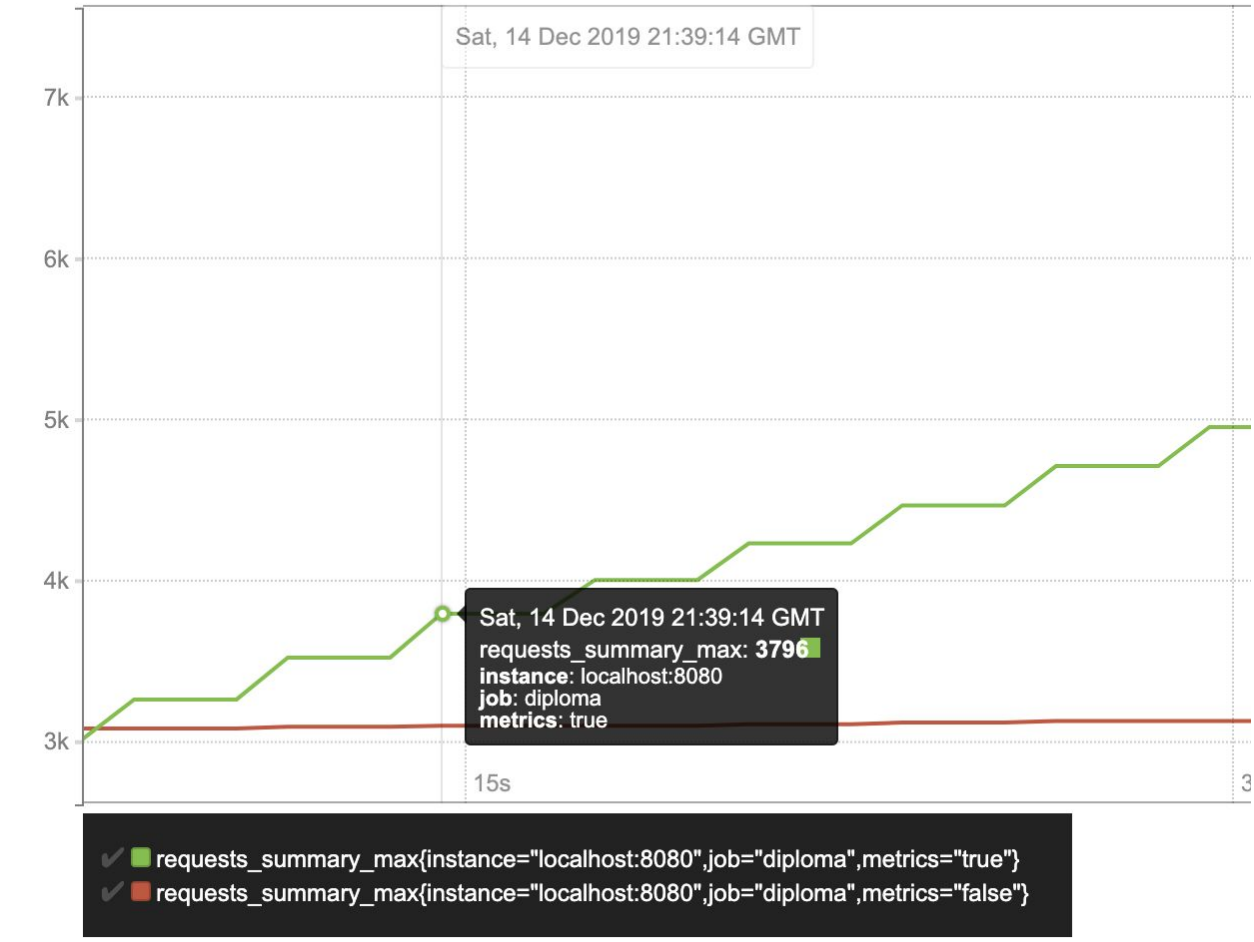


Рис. 3.13. Кількість обчислених проміжків

При використанні незмінної кількості проміжків волонтер опрацював дані, зробивши 25 запитів. Кількість заблокованих проміжків в такому випадку завжди дорівнювала кількості проміжків, що отримує волонтер, і

відповідає minAmount.

При використанні запропонованої формули з коефіцієнтом та постійним значенням *minAmount* – 250 проміжків було опрацьовано за 12 запитів при штучній затримці на обчислення проміжків у дві секунди.

Також було проведено моделювання з використанням статистичних даних. Було проведено декілька сесій обчислення з встановленим значенням штучної затримки в 100 секунд.

При ігноруванні зібраних метрик за попередні сесії обчислення, серверна частина відразу почала повертати користувачу *minAmount*.

При використанні вже зібраних метрик, система повертала користувачеві відразу тільки 1 проміжок, як мінімальну можливу кількість транспортованих даних, за рахунок чого вдалося не блокувати додаткові 9 проміжків на довгий проміжок часу.

Використання штучних затримок значно спростило тестування, а також надало можливість відстежувати зміну коефіцієнту (2.2). При значному збільшенні значення штучної затримки кількість проміжків теж значно зменшилась, і навпаки, при встановленні значення в початкове значення, система повернулась до попереднього значення кількості проміжків.

РОЗДІЛ 4. ТЕСТУВАННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

4.1. Ручне тестування

Для безпечного внесення змін в програмний продукт, він повинен бути покритий автоматичними тестами, які засвідчують коректність роботи застосунку. Такі тести надають змогу бути впевненим, що внесення змін не спричиняє появу помилок при виконання коду.

Було розроблено набір функціональних тестів для перевірки коректності виконання програми до внесення змін. Після покриття існуючих функціональних можливостей вирішено працювати надалі по методології TDD (Test Driven Development).

Розробка через тестування – технологія розробки програмного забезпечення, яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести. Варто зауважити, що керована тестами розробка є методологією розробки програмного забезпечення, а не його тестування. Використання даної методики надало можливість значно легше вносити зміни в існуючу функціональність, а також допомагало слідкувати за архітектурою розблюваної системи – збереження модульності.

Для полегшення подальшого моделювання та перевірки гіпотез було розроблено набір інтеграційних тестів, що перевіряли комунікацію клієнтської та серверної частин. Для перевірки клієнтської частини було вирішено використати набір заздалегідь підготовлених проміжків, а для серверної частини – набір заздалегідь підготовлених результатів.

Для проведення якісного тестування не можна було обійтись без використання браузера як засобу виконання програмного коду в системі розподілених обчислень.

Для початку було сформовано список тестових сценаріїв, які необхідно перевірити після внесення глобальних змін в програмний код системи. Дані тестові сценарії повинні були містити підготовлені набори вхідних даних та результати для перевірки. Доволі швидко кількість таких сценаріїв стала доволі великою і перешкоджала швидкому внесенню змін в програмний код, а також займала багато часу для проведення тестування, хоча й була дуже наглядною.

4.2. Автоматизоване тестування

4.2.1. Використання технології Cypress.js

Було вирішено застосувати засоби автоматизації тестів, в яких використовуються браузер. Сучасними засобами є бібліотеки Selenium та Cypress.js. Selenium – технологія, що використовує мову програмування Java та має власний програмний інтерфейс для проведення визначених тестових сценаріїв. Cypress.js – нова бібліотека мови JavaScript, що надає можливість також визначати тестові сценарії за допомогою власного програмного інтерфейсу, але надає значно більше можливостей для роботи з браузером під час виконання тесту у випадку знаходження помилок. Після аналізу переваг та недоліків було вирішено обрати технологію мови JavaScript для проведення автоматичного тестування в середовищі браузера.

Всі створені тестові сценарії, що до цього виконувались вручну, було запрограмовано. В результаті цього з’явилась можливість значно швидше вносити зміни в існуючий програмний код і перевіряти стан системи для проведення обчислень.

Програмування цих тестових сценаріїв базувалось на визначенні порядку необхідних переходів між сторінками та натисканні необхідних кнопок, завдяки чому відбувалось створення сесії обчислення. Так як бібліотека використовує справжній браузер, то середовище виконання коду в тестових сценаріях повністю співпадає з середовищем користувача. Це надає можливість знайти виключні ситуації значно раніше і бути впевненим в працездатності створеного рішення. Бібліотека надає власний програмний інтерфейс для роботи з браузером, а також можливість перевизначати відповіді запитів на серверну частину, отримувати детальну інформацію про відправлені запити та отримані відповіді.

Також дуже доречною є можливість ввімкнення та вимкнення підтримки веб-воркерів бібліотекою, що дало можливість провести тестування обчислення із залученням веб-воркерів та без них. В середовищі виконання тестових сценаріїв нам не важливе блокування взаємодії браузера з користувачем, перевірка правильного асинхронного виконання коду без веб-воркерів надало можливість знайти декілька виключних ситуацій в написаному програмному коді.

Доволі швидко стало очевидним, що дане рішення погано масштабується: швидкість виконання тестів була повільною, адже вони повністю залежали від швидкості роботи браузера. При цьому багато часу витрачається на відкриття бібліотекою вікна браузера, очікування повного завантаження сторінки. Дані кроки необхідно було повторювати й очікувати під час виконання кожного тестового сценарію. Ще одним обмеженням є послідовне виконання тестів, що також негативно впливає на швидкодію.

4.2.2. Використання технології Puppeteer

Оскільки на швидкодію виконання коду найбільше впливає саме необхідність відображення браузера бібліотекою, було вирішено провести аналіз рішень, які надають можливість виконувати тестові сценарії без відображення вікна браузера.

Бібліотека Puppeteer надає можливість користуватись браузерами на рушії Chromium без залучення інтерфейсної оболонки, обмежуючись тільки програмним інтерфейсом. Бібліотека написана мовою програмування JavaScript для популярного серверного фреймворку Node.

Особливістю бібліотеки є велика кількість зручних абстракцій, які надають можливість рухатись сторінками веб-ресурсу, натискати кнопки, створювати з'єднання та комунікувати з серверною частиною, викликаючи програмні методи. Бібліотека самостійно очікує завантаження сторінок та їх умовне відображення. Незважаючи на відсутність інтерфейсу, бібліотека все рівно потребує певного відображення, що в термінології браузера називається рендерингом. Без проведення рендерингу буде неможливим виконувати звичайні дії, такі як натискання кнопок.

Незважаючи на цей недолік, виконання тестових сценаріїв, використовуючи бібліотеку Puppeteer, стало значно швидшим і надало можливість ще швидше вносити зміни в програмний код для перевірки подальших гіпотез, пов'язаних із застосуванням формул. Дуже важливою особливістю бібліотеки виявилась можливість перехоплювати всі з'єднання для додаткового прослуховування. Викликавши декілька програмних методів бібліотеки, мережеві запити та відповіді не тільки потрапляли в браузер та серверну частину, а також могли бути продубльовані (наприклад в локальний файл файлової системи). Це дало можливість відстежувати комунікацію між

клієнтською та серверною частинами, прибравши значні шматки програмного коду, які було написані для зручності розробки.

Останнім недоліком так і залишилось послідовне виконання тестів, що було обумовлене тепер використанням певного порту. Бібліотека не підтримує виконання тестових сценаріїв паралельно. Після аналізу даної ситуації було вирішено використовувати поширений метод паралелізації виконання певного завдання за допомогою виконання коду в різних процесах операційної системи. Таким способом реалізується паралелізм багатьох сучасних інтерпритованих мов програмування та бібліотек, що базуються на них.

Принцип ґрунтується на розмежуванні відповідальності: програмний застосунок не передбачає виконання свого програмного коду багатьма потоками, цим займається розробник під час запуску програмного застосунку.

За допомогою інструкцій командного рядку програма запускається в багатьох різних процесах, таким чином операційна система виконує декілька екземплярів програми одночасно. Таким чином можна виконувати декілька тестових сценаріїв одночасно. Даний спосіб не можна було використовувати з попередніми бібліотеками, адже вони потребували відкриття інтерфейсної оболонки браузера.

Дане рішення потребувало вирішення проблеми зайнятого порту. Кожен тестовий сценарій запускається у власному процесі, а отже й потребує окремий порт для комунікації з серверною частиною. Використання одного порту є неможливим, адже операційна система повертає помилку про те, що даний порт вже використовується іншим процесом. Також виникла проблема занадто великого навантаження на систему, адже створювалась велика кількість процесів, що не були обмежені у використанні ресурсів.

Очевидно, що дана проблема вже не раз виникала в розробників під час вирішення доволі тривальних задач, тому було прийнято рішення проаналізувати проблему і знайти найпростіше рішення.

4.2.3. Контейнеризація

Використання віртуальних машин відразу було відкинуто через потребу у значних ресурсах для проведення тестування, значних витрат часу на налаштування інфраструктури, а також відсутності необхідності у самостійних віртуальних операційних системах для запуску єдиного процесу. Тому було вирішено використовувати можливості операційної системи – контейнеризацію.

Контейнеризація надає можливість запускати процеси незалежно один від одного за допомогою можливостей операційної системи. Кожен контейнер має власну область імен, процесів, а також виділені ресурси, які можна обмежувати. Також кожен контейнер має власну мережеву сітку та інтерфейс, що надає можливість виділяти порти незалежно від інших контейнерів.

Головною ідеєю було виконання тестового сценарію в контейнері, що буде комунікувати з серверною частиною, запущеною на основній операційній системі. Після виконання тесту всі необхідні результати потраплятимуть в файл з назвою тестового сценарію на основній операційній системі.

Для роботи з контейнерами було вирішено використовувати технологію Docker. Незважаючи на інші засоби, що надають ідентичні можливості, Docker зарекомендував себе як перевірений часом застосунок з великої кількістю користувачів.

Особливістю Docker є створення контейнерів на основі знімків, які описуються власною мовою. За допомогою обмеженого набору команд створюється знімок, де вказується середовище виконання коду, певні підготовчі процеси перед виконанням коду, а також процес, що буде виконуватись після створення контейнеру.

Особливістю технології Docker є також використання вже готових знімків для більшості існуючих технологій та бібліотек. В мережі було знайдено вже налаштований офіційний знімок технології Puppeteer, який потребував тільки налаштування застосунку, що буде тестуватись.

В результаті було отримано контейнер, який отримував назву тестового сценарію в якості вхідного параметру, виконував вказаний тестовий сценарій та зберігав всі результати виконання в локальний файл на операційній системі. Так як кожен контейнер має власну мережну сітку, необхідний порт залишався вільним, що дало можливість виконувати тестові сценарії в різних контейнерах одночасно.

Docker надає можливість вказувати обмеження для ресурсів, що будуть використовуватись контейнером, тому проблема навантаження на систему розробника також була вирішена.

Для запобігання написання власних скриптів команд для запуску багатьох контейнерів з тестовими сценаріями одночасно, було вирішено використати допоміжну технологію під назвою Docker stack, яка є частиною Docker. Вона надає можливість запускати більше ніж один контейнер одночасно та вказувати їм різні вхідні параметри.

4.3. Висновки

В результаті аналізу великої кількості технологій та різних підходів до тестування було знайдено найбільш доцільне рішення проблеми тестування серверно-клієнтських застосунків.

Застосування контейнеризації надало можливість моделювати проведення обчислень в розподілених волонтерських системах, використовуючи одну інструкцію в командному рядку операційної системи.

Як наслідок, вдалося на одній системі організувати паралельні тестові сценарії, які виконували моделювання процесів волонтерських обчислень у браузері з використання різних формул для обчислення кількості проміжків даних для транспортування. Таке моделювання значно спрощує перевірку гіпотез використання формул для зменшення кількості запитів на серверну частину.

РОЗДІЛ 5. ПОБУДОВА БІЗНЕС МОДЕЛІ

5.1. Аналіз проблеми

Покращення в області проведення довготривалих обчислень може сприяти розвитку багатьох суміжних та залежних областей. Для пришвидшення обчислень та збільшення об'єму даних для обчислень використовується вертикальне та горизонтальне масштабування систем. Розподілені волонтерські обчислення за допомогою браузеру є доволі перспективним рішенням, враховуючи поширеність пристроїв з підключенням до мережі Інтернет.

Але, вирішуючи проблеми масштабованості систем із великими вхідними даними, розподілені волонтерські обчислення породжують низку нових питань, до яких входять проблеми безпеки та довіри, швидкодії та своєчасності отримання результатів.

Виконання розподілених обчислень на серверній частині практично нівелює питання швидкодії обчислень, але втрачається масштабованість обчислень.

Існуючі системи, такі як BOINC, впроваджують розподілені волонтерські обчислення в наукових галузях та не мають комерційного характеру. Розроблення системи проведення обчислень загального характеру надасть можливість проводити хмарні обчислення на потреби бізнесу.

На сьогоднішній день, для виконання завдань пов'язаних з інформаційними технологіями створюються комплекси обчислювальні системи. Написання відповідних систем (систем, що потребуються довготривалих обчислень, або систем, що працюють з великими обсягами даних) є складним завданням, а також потребує великих фінансових ресурсів.

Використання хмарних технологій для проведення обчислень в значній мірі спрощує процес налаштування інфраструктури, але є також потребує значних фінансових ресурсів, адже проведення обчислень все рівно відбувається на серверних машинах провайдера хмарних послуг.

Зрозуміло, що для великих корпорацій результат отриманий після проведення обчислень є дуже цінним і повинен бути отриманий якнайраніше, тому вкладення значних фінансових ресурсів є доцільним. Але для бізнесу малого (та можливо середнього) рівня набагато важче витрачати ресурси на ще не перевірені галузі отримання прибутку, тому вони потребують системи перевірки гіпотез через проведення обчислень.

Розроблювана система надає можливість саме проводити обчислення, робити довготривалу обробку великих обсягів даних, а не слугує безкоштовним варіантом приватного віртуального серверу для створення веб-сайтів для бізнесу. Система орієнтована на періодичне отримання вхідного набору даних та повернення вже обробленого результату, надає можливість проводити обмежений набір налаштувань, таких як перевірка коректності обчислень.

5.2. Дерево проблем

Базове дерево проблем проекту зображено на рис. 5.1.

Найголовніша проблема сучасного ринку є саме необхідність у великих фінансових витратах навіть для перевірки гіпотези. Проведення повноцінних обчислень вже потребує власних серверних потужностей або витрат на хмарні потужності.

Рис. 5.1. Базове дерево проблем

Існуючі системи проведення розподілених обчислень більше пристосовані до проведення наукових обчислень та потребують встановлення додаткових застосунків, що значно зменшує кількість активних користувачів-волонтерів. Також більшість існуючих застосунків ігнорують наявність підключених до мережі великої кількості мобільних систем, є застарілими й перешкоджають поширенню волонтерських систем як явища (відбувається велика втрата можливих користувачів-волонтерів).

Також не варто ігнорувати доцільність використання інформаційних технологій для вирішення питань бізнесу. Саме завдяки популярності та доцільності й з’явилась велика кількість ресурсів, що надають змогу залучати користувачів також в мережі Інтернет, без самостійного створення комплексного рішення. Ресурси-конструктори Інтернет сторінок використовуються також через відсутність необхідності займатись інфраструктурою рішення, витрачати кошти на розробників та підтримку рішення.

Відсутність на ринку рішення для проведення обчислень таким самим чином, як створення веб-ресурсів, змушують користувачів знаходити альтернативні та менш оптимальні рішення підтвердження гіпотез.

Також гостро стоїть проблема необхідності тільки періодичних обчислень. У випадку витрат на власні обчислювальні потужності, вони виглядають недоцільними, адже не відбуватиметься утилізація необхідного рівня. Також не зважаючи на можливості сучасних хмарних обчислювальних систем, їх налаштування є складним процесом, а утилізація є доволі дорогою при недостатньому користуванні всіх потужностей.

Сучасні технології безсерверних обчислень також мають доволі високу ціну та недостатню популярність серед розробників через незрілість пропонованого рішення.

5.3. Зацікавлені сторони

У вирішенні описаної проблеми існує декілька зацікавлених сторін.

Найбільш очевидною та впливовою зацікавленою стороною є саме малий бізнес, що потребує перевірки певних гіпотез, пов’язаних з інформаційними технологіями, та проведення періодичних обчислень. В даної групи відсутня можливість витрачати великі фінансові ресурси для проведення обчислень, тому наявність системи, що надасть можливість періодично отримувати цінні дані без залучення значних зусиль є дуже привабливою.

Також зацікавленою стороною є саме користувачі-волонтери, але їх зацікавленість базується саме на особистих переконаннях. Також більшість сучасних систем проведення волонтерських обчислень зосереджені на проведенні обчислень наукового характеру та все ще потребують встановлення додаткових застосунків, що наявні тільки для персональних

комп’ютерів та ноутбуків. Веб-ресурс для проведення обчислень значно збільшить кількість активних користувачів-волонтерів через спрощення системи.

Приваблювати користувачів-волонтерів також планується завдяки “гейміфікації” процесу проведення обчислень, тобто створення в користувачів бажання змагатися між собою. Основним засобом зацікавлення користувачів є система досягнень.

Зацікавленою стороною є також “провайдери” хмарних обчислювальних потужностей, адже їм доведеться залишатися конкурентноспроможними на ринку проведення періодичних довготривалих обчислень. Також не потрібно забувати, що система розподілених волонтерських обчислень також потребує наявності сервених потужностей для проведення синхронізації обчислень та роботи зі з’єднаннями користувачів. Замість навантаження серверних машин відбудеться навантаження саме мережевих комунікаційних систем.

Також зацікавленою стороною є наукові товариства, що готові проводити обчислення в розподілених волонтерських платформах. Досвід показав (система BOINC), що використання розподілених волонтерських платформ надає можливості проводити обчислення без значних витрат завдяки зацікавленим волонтерам. Наразі система BOINC займається проведенням обчислень для відслідковування руху космічних об’єктів і так далі. Створення системи проведення розподілених волонтерських обчислень без необхідності встановлення додаткових застосунків значно збільшить кількість волонтерів, а отже й збільшить пропускну здатність платформи.

Таблиця 5.1

Зацікавлені сторони

Зацікавлена сторона	Інтерес зацікавленої особи	Вплив зацікавленої особи	Стратегії приваблення
Малий бізнес	Проведення періодичних довготривалих обчислень без необхідності залучення великих фінансових ресурсів.	Високий	Дуже низькі витрати на проведення обчислень.
Користувачі-волонтери	Особисті переконання в необхідності проведення волонтерських обчислень.	Середній	Спрощення системи та застосунків, використання вже існуючих застосунків для проведення обчислень.
Наукові товариства	Проведення наукових обчислень без залучень додаткових коштів, перевірка гіпотез.	Середній	Надання значно більшої бази активних волонтерів, а отже і обчислювальних потужностей.
Провайдери	Отримання статистичних даних, покращення власних систем.	Низький	Конкуренція змусить сторону створювати кращі умови для користувачів.

5.4. Опис продукту

Результатом дослідницької роботи буде програмне забезпечення, але слід зазначити, що ПЗ не основний виробничий продукт, в основі ПЗ лежить оптимізація процесу транспортування даних між серверною та клієнтськими частинами.

ПЗ надасть можливість проводити волонтерські обчислення, створюючи в системі завдання та завантажуючи необхідні дані. Після проведення обчислень, користувач отримує результати певним вибраним чином.

Продукт являє собою систему з серверної частини, що займається синхронізацією обчислення між користувачами та збереженням даних, та клієнтські веб-застосунки для звичайного користувача-волонтера та адміністратора відповідно. Веб-застосунок для адміністратора надає можливість створювати та налаштовувати завдання для обчислення. Також є можливість отримати результати обчислення в довільній формі. Застосунок для користувача-волонтера надає можливість приєднюватись до обчислення певного завдання, а також слідкувати за його станом. Система також збирає певні метрики для покращення швидкодії та надійності обчислень. Дані метрики використовуються для покращення методу транспортування даних між серверною і клієнтською частинами.

Також продукт надає можливість робити велику кількість налаштувань для проведення обчислення, а також для обробки отриманого результату. Так як дана система побудована по принципу “MapReduce”, для завершення виконання обчислення необхідно виконати два кроки: провести обчислення кожного елементу вхідного набору даних, а також провести агрегацію отриманих результатів. Кожен з цих кроків може бути окремо налаштований

під певні потреби користувача. Наприклад, крок агрегації підтримує різні формати отримання результатів обчислення.

Також систему може бути розгорнуто в будь-якій закритій мережі для створення кластеру для обчислення без врахування фактору волонтерства, хоча ефективність обчислень стрімко знизиться у порівнянні з іншими існуючими застосунками.

Бізнес-рішенням є монетизація створення завдання в системі в залежності від розміру набору вхідних даних та вибраних налаштувань (таких як перевірка коректності обчислення).

5.4.1. Ціннісна пропозиція продукту

Цінність – це розуміння користі, яку покупець отримує від продукту. З такої позиції можна чітко охарактеризувати цінність продукту для кожної зацікавленої сторони, або для кожного потенційного користувача, агрегувати ці відомості та побудувати детальну інформацію.

Так як цінність є суб’єктивною величиною, то далі наведена цінність відносно зацікавлених осіб.

- Малий бізнес
    - Можливість перевірки гіпотез пов’язаних з інформаційними технологіями. Під перевіркою гіпотез мається на увазі використання рішень пов’язаних з інформаційними технологіями для вирішення бізнес проблем. Це може включати обробку певних збережених даних про клієнтів, обрахунок статистики, навіть надсилання електронних листів.
    - Дешеві періодичні обчислення. Відсутність необхідності витрачати лишні фінансові ресурси на розробників, серверні обчислювальні потужності. Планується розробка

системи-конструктору, що надаватиме можливість створювати примітивні операції без знання мов програмування взагалі.

- Відсутність постійних витрат на інформаційні технології. Витрати з’являються тільки у випадку активного використання системи, але витрачаються фінанси тільки на ефективні обчислення.

- Користувачі-волонтери
  - Задоволення власних потреб, пов’язаних з волонтерством. Спираючись на досвід вже існуючих систем розподілених волонтерських обчислень, позиція волонтерства гостро стоїть в суспільстві, люди готові надавати обчислювальні ресурси для вирішення чужих проблем. Але зазвичай ці проблеми пов’язані з науковими обчисленнями, комерційні обчислення поки що не були перевірені готовими рішеннями.
- Наукові товариства
  - Можливість проведення довготривалих обчислень без залучення фінансових ресурсів. Спираючись на доволі велику кількість наукових проектів, що використовують системи розподілених волонтерських обчислень, можна сміливо стверджувати, що попит на такі системи є.

5.4.2. Бізнес-модель

Бізнес-модель – це те, як ми організовуємо нашу діяльність, щоб надати замовнику обіцяну цінність. Це те, як ми організовуємо наші внутрішні операції, щоб виробляти і доставляти цінність замовнику.

1. Ключовими партнерами є зацікавлені сторони.

2. Ключовою діяльністю є забезпечення проведення розподілених волонтерських обчислень за допомогою веб-застосунку.
3. Ціннісна пропозиція є забезпечення можливості проведення періодичних довготривалих обчислень для зацікавлених сторін, а також забезпечення задоволення волонтерських потреб користувачів-волонтерів.
4. Клієнтським сегментом є наукові товариства, що оперують великими обсягами даних, а також малий бізнес, що потребує періодичних обчислень.
5. Ключовими ресурсами є сервери для синхронізації розподілених обчислень.
6. Канали збуту: веб застосунки.
7. Ціннісна структура – це система проведення розподілених волонтерських обчислень (серверна та клієнтська частини).

В табл. 5.2 наведено канву запропонованої бізнес моделі.

Таблиця 5.2

Канва бізнес-моделі

<table>
<tr>
<td rowspan="2">Проблема

Відсутність рішень для проведення комерційних розподілених обчислень; висока вартість власних та хмарних обчислювальних потужностей; відсутність систем для проведення тільки періодичних обчислень.</td>
<td>Рішення

Програмне забезпечення, що надає можливість проводити періодичні розподілені волонтерські обчислення без значних витрат ресурсів.</td>
<td rowspan="2">Унікальна ціннісна пропозиція

Зменшення витрат на перевірку гіпотез; відсутність необхідності завантаження застосунків.</td>
<td>Прихована перевага

Оптимізація обчислення через врахування користувацьких метрик.</td>
<td rowspan="2">Споживачі

Малий бізнес, який бажає долучитися до інформаційних технологій; наукові товариства; волонтери.</td>
</tr>
<tr>
<td>Ключові метрики
Кількість активних користувачів-волонтерів; кількість обчислювальних завдань в системі.</td>
<td>Канали

через рекламу на спеціальних ресурсах.</td>
</tr>
<tr>
<td colspan="2">Структура витрат

Витрати на персонал підтримки системи;
Витрати на податки;
Витрати на серверні потужності.</td>
<td colspan="3">Потоки доходів

Дохід від створення завдання для обчислення в системі.</td>
</tr>
</table>

5.5. Висновки

На основі проведеного дослідження можливо побудувати працюючу бізнес-схему.

Основною цінністю даного продукту є можливість проведення розподілених волонтерських обчислень для наукових товариств та зацікавлених користувачів.

Розроблений програмний продукт має джерела прибутку у вигляді монетизації створення завдання в системі та канали збуту.

Було проведено аналіз майбутніх клієнтів. У результаті була описана бізнес-модель, що обґрунтовує доцільність реалізації даного продукту та прогнозує його потенційну окупність та прибутковість в подальшому.

ВИСНОВКИ

У даній магістерській роботі виконано наступні завдання:

- проведено аналіз існуючих методів визначення обсягу даних для транспортування, вказано особливості їх використання, визначено переваги та недоліки;
- досліджено способи отримання (за сесію та збережених) поведінкових особливостей користувача за допомогою програмного інтерфейсу браузера (виміри тривалості виконання, використання cookies) та їх збереження;
- поведінкові особливості користувачів формалізовано у вигляді набору метрик;
- запропоновано формули обчислення обсягу вхідних даних для транспортування із застосуванням метрик на основі поведінки користувача, його параметрів конфігурації та даних серверної частини;
- отримані формули застосовано у програмному забезпеченні для проведення волонтерських розподілених обчислень;
- процес моделювання проведення волонтерських обчислень автоматизовано за допомогою контейнеризованого паралельного тестування.

Результати, отримані при реалізації способу, показали, що він надає можливість зменшити кількість запитів на серверну частину для кожного користувача індивідуально.

Найкращі результати отримано при використанні метрики активної сесії користувача та статистичних метрик для визначення початкового обсягу вхідних даних.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. BOINC: A System for Public-Resource Computing and Storage [Text] / David P. Anderson. – Pittsburgh: USA. – November 8, 2004.
2. Volunteer computing: The ultimate cloud [Text] / D. P. Anderson. – ACM Crossroads, vol. 16, no. 3, Mar. 2010. – pp. 7-10.
3. Designing a Runtime System for Volunteer Computing [Text] / David P. Anderson, Carl Christensen, Bruce Allen. – Tampa. – November 2006.
4. High-Performance Task Distribution for Volunteer Computing [Text] / David P. Anderson, Eric Korpela, Rom Walton. – Melbourne. – 5 December 2005.
5. A Comparison of Techniques for Distributing File-based Tasks for Public-Resource Computing. Proceedings of The 17th IASTED International Conference on Parallel and Distributed Computing and Systems – PDCS 2005 [Text] / D. Toth, D. Finkel. – Phoenix : Arizona, USA. 2005. – pp. 398-403.
6. Security Threats in Volunteer Computing Environments Using the Berkeley Open Infrastructure for Network Computing (BOINC) [Text] / Paulo Picota Cano, Miguel Vargas-Lombardo. – Volume 7. – pp. 727–734.
7. ComcuteJS: A Web Browser Based Platform for Large-scale Computations [Електронний ресурс] – Режим доступу https://www.researchgate.net/publication/259932102_ComcuteJS_A_Web_Browser_Based_Platform_for_Large-scale_Computations – Дата доступу: 05.03.2019 – Назва з екрану.
8. Sabotage-tolerance mechanisms for volunteer computing systems [Text] / Sarmenta. – 2002. – pp.561-572

ДОДАТКИ

Додаток 1

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Спосіб знаходження обсягу вхідних даних для транспортування в системах розподілених волонтерських обчислень

Виконав: Орос Богдан Богданович

Науковий керівник: старший викладач, к.т.н. Рибачок Наталія Антонівна

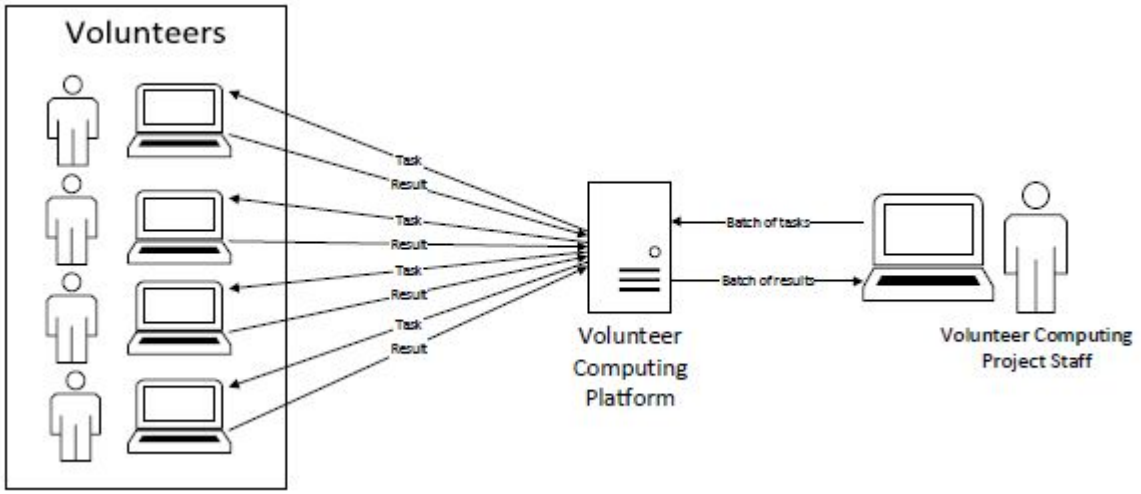
Київ – 2019

АКТУАЛЬНІСТЬ ДОСЛІДЖЕННЯ СПОСОБУ ЗНАХОДЖЕННЯ ОБСЯГУ ВХІДНИХ ДАНИХ

Більше ніж 30 активних проектів наукового характеру на BOINC:

- Acoustics@home
- Asteroids@home

Волонтерські обчислення – тип розподілених обчислень, де користувач (власник комп'ютеру, котрий знаходиться в мережі) надає обчислювальні потужності власного комп'ютеру для виконання обчислень.



Транспортування даних – переміщення вхідних даних завдання для обчислення за допомогою мережевих запитів з серверної на клієнтську частину.

Об’єкт та предмет дослідження

Об’єкт дослідження: процес знаходження обсягу вхідних даних для транспортування ґрунтуючись на поведінкових особливостях волонтера.

Предмет дослідження: способи знаходження обсягу вхідних даних для подальшого транспортування.

Мета дослідження

Розробити спосіб знаходження обсягу проміжку вхідних даних для транспортування в системах розподілених волонтерських обчислень.

Постановка науково-технічної задачі

1. Провести аналіз існуючих методів визначення обсягу даних для транспортування.
2. Дослідити способи отримання поведінкових особливостей користувача.
3. Формалізувати отриманні поведінкові особливості користувачів у вигляді метрик.
4. Застосувати вибрані метрики для виведення формул обчислення обсягу вхідних даних для транспортування.
5. Застосувати отримані формули у програмному забезпеченні для проведення волонтерських розподілених обчислень.

Дослідники

- Roman Dębski, Tomasz Krupa, Przemysław Majewski. ComcuteJs. 2013
- Sean Wilkinson. QMachine. 2014

Існуючі способи

- Використання проміжків одного розміру
- Збільшення розміру проміжку в залежності від тривалості сесії обчислення
- Ручне введення необхідних параметрів

Використання проміжків одного розміру

Переваги:

- Простота розробки та підтримки
- Передбачуваність

Недоліки:

- Відсутність гнучкості.
- Надлишкові запити до сервера.

Збільшення розміру проміжку в залежності від тривалості сесії обчислення

Переваги:

- Простота реалізації
- В кращих випадках дає хороший приріст в швидкодії

Недоліки:

- Залежність від часу
- Велика кількість заблокованих проміжків

Ручне введення необхідних параметрів

Переваги:

- Простота реалізації
- Висока пропускна здатність

Недоліки:

- Необхідність довіри до клієнтів

Запропоновані метрики

- Кількість обчислених проміжків за сесію.
- Кількість обчислених проміжків в середньому за всі існуючі сесії користувача (якщо користувача вдалось ідентифікувати).
- Швидкість обчислення проміжку за сесію.
- Середня швидкість обчислення проміжку за всі існуючі сесії користувача (якщо користувача вдалось ідентифікувати).
- Відношення правильних результатів обчислення до помилкових.
- Встановлений користувачем рівень навантаження на систему.
- Інформація від браузера про апаратні можливості користувача.

Приклад використання метрик

$$r = minAmount * (1 + \varepsilon_n) * a_n,$$

$$\varepsilon_n = n / \sum_{i=0}^{n} t_i$$

$$a_n = \varepsilon_n / \varepsilon_{n-1}$$

n – кількість проміжків обчислених волонтером за сесію,

minAmount – мінімальна кількість проміжків, що надається волонтеру сервером,

ε_n – коефіцієнт для визначення продуктивності системи волонтера.

Приклад використання метрик

$$r = M(N) * (1 + \varepsilon_n) * a_n,$$

де N – кількість обчислених проміжків кожною з сесій обчислення.

Приклад використання метрик

$$r = g(i) * (1 + \varepsilon_n) * a_n$$

де i – унікальний стандартизований ідентифікатор обчислювальної системи.

Приклад використання метрик

$$R = r \wedge maxAmount,$$

де *r* – кількість проміжків, отриманих в результаті обчислення попередніми формулами для визначення кількості проміжків,

maxAmount – максимальна кількість проміжків, які можуть бути транспортовані користувачу за один запит.

Моделювання

Вхідні дані для обчислень містили 250 проміжків,
minAmount=10.

При використанні незмінної кількості проміжків:

- волонтер опрацював дані, зробивши 25 запитів;

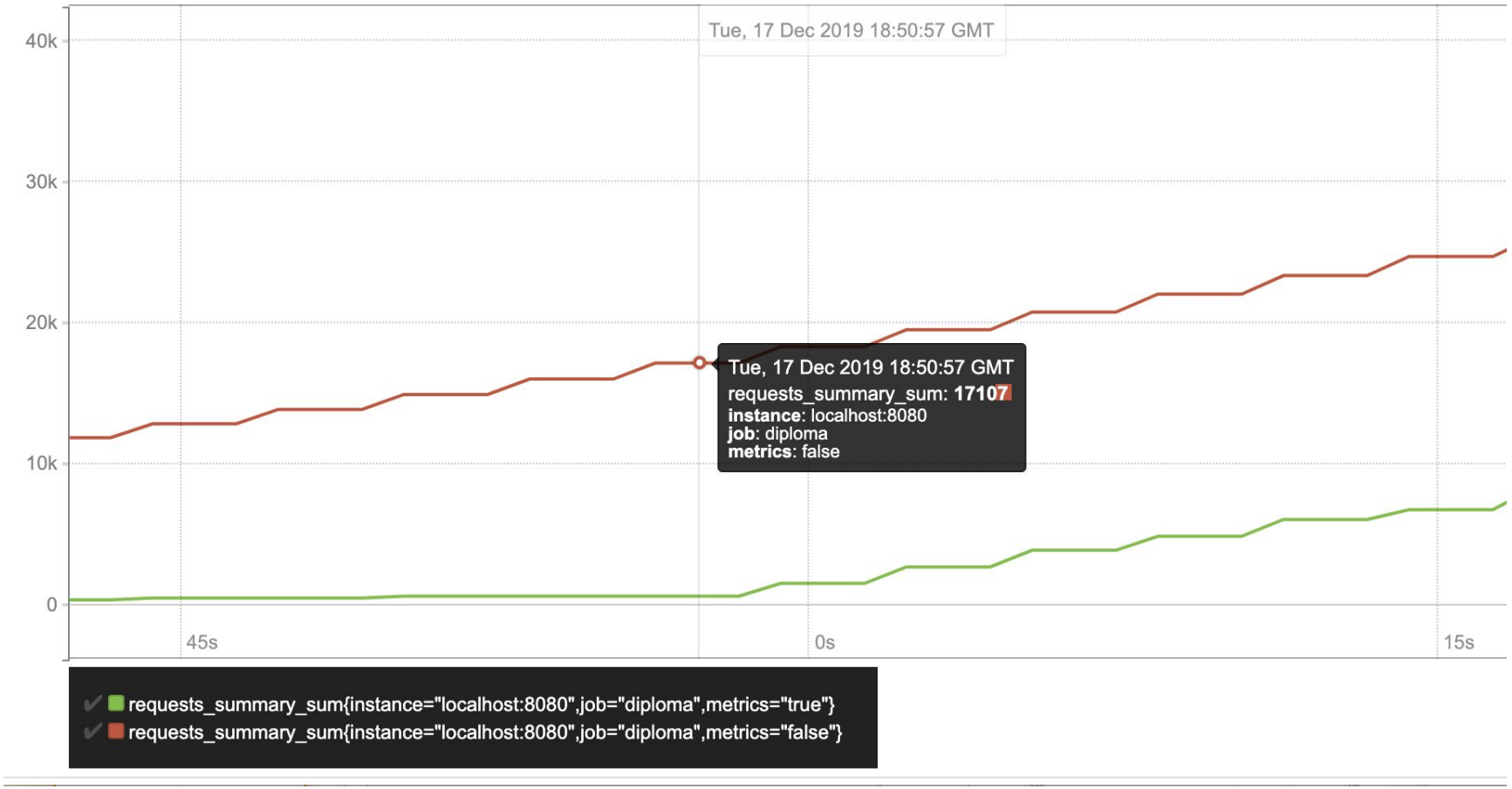
При використанні запропонованої формули:

- волонтер опрацював дані, зробивши 12 запитів при штучній затримці на обчислення проміжків у дві секунди;

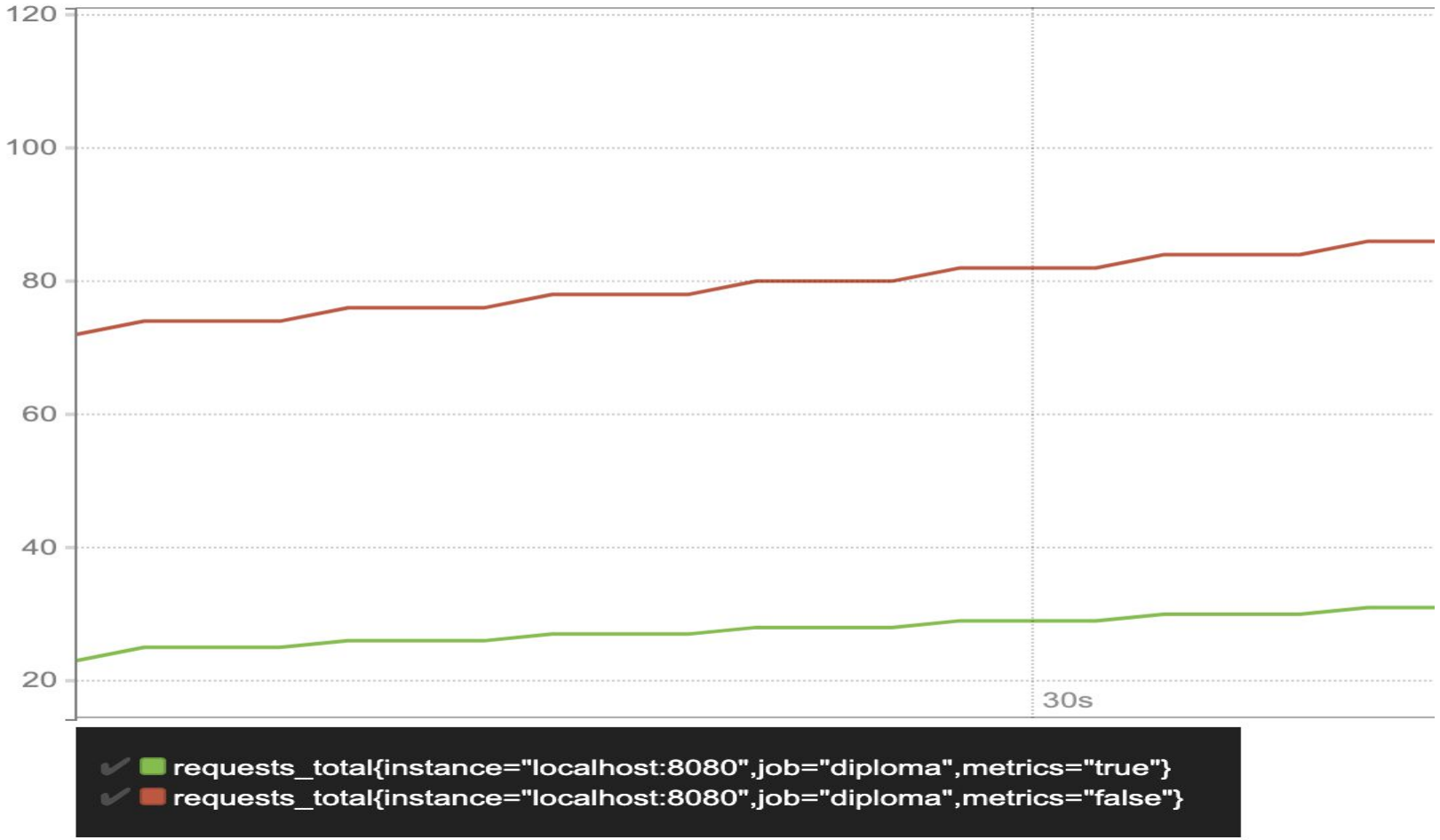
Моделювання

Виконання інтеграційних тестових сценаріїв за допомогою контейнеризації та технології Docker надало можливість швидше перевіряти гіпотези з використанням метрик.

Моделювання



Моделювання



НАУКОВО-ІННОВАЦІЙНА НОВИЗНА

- Запропоновано набір метрик, який доцільно використовувати для обчислення обсягу вхідних даних індивідуально під кожного користувача.
- Розроблено спосіб знаходження обсягу вхідних даних для транспортування з використанням метрик, таким чином зменшуючи навантаження на серверну частину способом зменшення кількості мережевих запитів.

ВИСНОВКИ

1. Проведено аналіз існуючих методів визначення обсягу даних для транспортування, вказано особливості їх використання, визначено переваги та недоліки.
2. Досліджено способи отримання (за сесію та збережених) поведінкових особливостей користувача за допомогою програмного інтерфейсу браузера (виміри тривалості виконання, використання cookies) та їх збереження.
3. Поведінкові особливості користувачів формалізовано у вигляді набору метрик.
4. Запропоновано формули обчислення обсягу вхідних даних для транспортування із застосуванням метрик на основі поведінки користувача, його параметрів конфігурації та даних серверної частини.

ВИСНОВКИ
(продовження)

5. Отримані формули застосовано у програмному забезпеченні для проведення волонтерських розподілених обчислень.
6. Процес моделювання проведення волонтерських обчислень автоматизовано за допомогою контейнеризованого паралельного тестування.

АПРОБУВАННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

1. Наукова конференція магістрантів та аспірантів «Прикладна математика та комп’ютинг» (ПМК-2019-2).

Дякую за увагу!

Додаток 2

Приклади програмного коду

Runner.java – файл обробки запитів і визначення кількості проміжків.

```
package com.boros;

import com.boros.task.InputData;
import com.boros.task.RunningTask;
import com.boros.task.Task;
import com.boros.task.TaskRepository;
import com.google.gson.Gson;
import io.micrometer.core.instrument.Counter;
import io.micrometer.core.instrument.DistributionSummary;
import org.eclipse.jetty.websocket.api.Session;
import org.eclipse.jetty.websocket.api.annotations.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.atomic.AtomicLong;

@WebSocket
public class RunningTaskHandler {

    private final Logger logger =
LoggerFactory.getLogger(RunningTaskHandler.class);

    private final Gson gson = new Gson();

    private final IdleSessionService idleSessionService = new
IdleSessionService();
    private final RunningTaskService runningTaskService = new
RunningTaskService();

    private final AtomicLong statsCollector = new AtomicLong();
    private final AtomicLong slicesCounter = new AtomicLong();

    private final Map<String, Long> calculatedSlicesPerUser = new HashMap<>();
    private final Map<String, Double> calculationTimeInSeconds = new
HashMap<>();
    private final AtomicLong requestsCounter = new AtomicLong();
    private double prev;
    private long start;
    private long end;
```

```
    private Counter counter = Runner.prometheusRegistry.counter("requests",
"metrics", "true");
    private DistributionSummary summary =
Runner.prometheusRegistry.summary("requests_summary", "metrics", "true");

    private Counter counterWithoutMetrics =
Runner.prometheusRegistry.counter("requests", "metrics", "false");
    private DistributionSummary summaryWithoutMetrics =
Runner.prometheusRegistry.summary("requests_summary", "metrics", "false");


    private int minAmount = 10;

    @OnWebSocketConnect
    public void onConnection(Session session) throws IOException {
        idleSessionService.addSession(session);
        start = System.nanoTime();
        System.out.println(start);
        logger.info("New session connected. Session attached to idle sessions:
{}", session);
    }

    @OnWebSocketClose
    public void onClose(Session session, int statusCode, String reason) {
        logger.info("Session [{}] disconnected. Reason [{}]", session, reason);
        end = System.nanoTime();
        System.out.println(end);
    }


    @OnWebSocketError
    public void onError(Session session, Throwable throwable) {
    }


    @OnWebSocketMessage
    public void onMessage(Session session, String rawRequest) throws
IOException {
        // TODO 1) Create running task if not exists;
        // TODO 2) Attach session to running task;
        // TODO 3) Divide task input between users;
        // TODO 4) Send task input data part to user;
        Request request = gson.fromJson(rawRequest, Request.class);

        if (Action.ActionType.IDENTIFY.is(request.action)) {
            IdentifyRequest identifyRequest = gson.fromJson(rawRequest,
IdentifyRequest.class);
            String identity = identifyRequest.identity;
```

```
            if (identity == null) {
                String user = UUID.randomUUID().toString();
                IdentifyRequest resp = new IdentifyRequest(
                        Action.ActionType.IDENTIFY.type,
                        "identity=" + user + ";" + user);
                session.getRemote().sendString(gson.toJson(resp));
            }
        }

        if (Action.ActionType.ATTACH.is(request.action)) {
            Task task = TaskRepository.tasks.stream()
                    .filter(t -> t.id.equals(request.payload))
                    .findFirst()
                    .get();
            RunningTask runningTask =
runningTaskService.attachToRunningTasks(task, session);
            logger.info("Attached session to running task {}", runningTask);


            long length = request.identity != null
                    ? calculatedSlicesPerUser.getOrDefault(request.identity,
1L)
                    : 1L;

            session.getRemote().sendString(gson.toJson(new
AttachResponse<>(task, task.input.getData((int) length))));


        } else if (Action.ActionType.RESULT.is(request.action)) {
            ResultRequest resultPayload = gson.fromJson(rawRequest,
ResultRequest.class);
            Task task = TaskRepository.tasks.stream()
                    .filter(t -> t.id.equals(resultPayload.result.task))
                    .findFirst()
                    .get();


            calculatedSlicesPerUser.put(request.identity,
                    calculatedSlicesPerUser.getOrDefault(request.identity, 0L)
+ resultPayload.result.result.size());
            calculationTimeInSeconds.put(request.identity,
                    calculationTimeInSeconds.getOrDefault(request.identity,
0.0) + resultPayload.time);

            long calculatedSlicesPerSession =
calculatedSlicesPerUser.get(request.identity);
            double timeSpentPerSession =
calculationTimeInSeconds.get(request.identity);
```

```
            if (resultPayload.metrics) {
                summary.record(calculatedSlicesPerSession);
            } else {
                summaryWithoutMetrics.record(calculatedSlicesPerSession);
            }

            double coefficient = calculatedSlicesPerSession /
timeSpentPerSession;
            double acceleration = Math.min(coefficient / prev, 1);
            double count = minAmount * (1 + coefficient) * acceleration;
            prev = coefficient;

            long ceiledCount = (long) (Math.ceil(count));
            int limitedCount = ceiledCount > 100 ? 100 : (int) ceiledCount;

            if (!resultPayload.metrics) {
                limitedCount = minAmount;
                counterWithoutMetrics.increment();
            } else {
                counter.increment();
            }

            System.out.println("Part in overral " + coefficient);
            System.out.println("Limited " + limitedCount);
            System.out.println("Average per slice: " + ceiledCount);
            System.out.println("Response: " + resultPayload.time);


            List<InputData.DataNode<String>> dataNodes =
runningTaskService.nextResult(resultPayload.result, limitedCount);
            session.getRemote().sendString(gson.toJson(new
AttachResponse<>(task, dataNodes)));
            slicesCounter.addAndGet(dataNodes.size());
            requestsCounter.incrementAndGet();
            statsCollector.addAndGet(limitedCount);

        }
        System.out.println("Current stats: " + requestsCounter);
        System.out.println("Count stats: " + statsCollector);
    }

    public static class Action {

        public String action;

        public enum ActionType {
            ATTACH("attach"),
```

```
        RESULT("result"),
        IDENTIFY("identity");

        final String type;

        ActionType(String type) {
            this.type = type;
        }

        public boolean is(String action) {
            return this.type.equals(action);
        }

    }

}

public class Request {

    public String action;
    public String identity;
    public Long payload;

}

public static class IdentifyRequest {
    private final String action;
    private final String identity;

    public IdentifyRequest(String action, String identity) {
        this.action = action;
        this.identity = identity;
    }
}

public class ResultRequest {
    public String action;
    public Double time;
    public boolean metrics;
    public ResultPayload result;
}

public class ResultPayload {
    public Long task;
    public Map<String, String> result;
}

public static class AttachResponse<T> {
```

```
        private final String action = Action.ActionType.ATTACH.type;
        private final Task task;
        private final List<T> inputData;

        AttachResponse(Task task, List<T> data) {
            this.task = task;
            this.inputData = data;
        }

    }

    private class SlicesCountCalculator {
        private long calculatedSlicesPerSession;
        private double timeSpentPerSession;
        private double coefficient;
        private double count;

        public SlicesCountCalculator(long calculatedSlicesPerSession, double
timeSpentPerSession) {
            this.calculatedSlicesPerSession = calculatedSlicesPerSession;
            this.timeSpentPerSession = timeSpentPerSession;
        }

        public double getCoefficient() {
            return coefficient;
        }

        public double getCount() {
            return count;
        }

        public SlicesCountCalculator invoke(int startingPoint) {
            coefficient = calculatedSlicesPerSession / timeSpentPerSession;
            double acceleration = Math.min(coefficient / prev, 1);
            count = startingPoint * (1 + coefficient) * acceleration;
            return this;
        }
    }
}
```

Tasks.js – файл виконання обчислень на серверній частині.

```
let webSocket = new WebSocket("ws://" + location.hostname + ":" + location.port
+ "/execution");
let activeTaskFunc = null;
let input = null;
let func = null;

let slowRatio = 20;
let metrics = false;

function sendTask(taskId, taskName) {
    let payload = {
        action: 'attach',
        identity: getIdentity(),
        payload: taskId
    };
    webSocket.send(JSON.stringify(payload));
    console.log("Task chosen: " + taskId);
    showChoosenTask(taskName);
}

webSocket.onmessage = function (msg) {
    let data = JSON.parse(msg.data);
    if (data.action === 'identity') {
        document.cookie = data.identity;
    }
    if (data.action === 'attach') {
        buildFunctionToExecute(data);
        if (activeTaskFunc) {
            activeTaskFunc.call();
        }
    }
};

function buildFunctionToExecute(data) {
    input = data.inputData;
    const task = data.task;
    func = new Function("str", task.function.executionFunction);
    activeTaskFunc = processActiveTaskFunction(task);

}

webSocket.onclose = function () {
    console.log("Closed");
    webSocket = new WebSocket("ws://" + location.hostname + ":" + location.port
+ "/execution");
```

```
};

webSocket.onopen = function (msg) {
    console.log("Opened");
    webSocket.send(JSON.stringify({
        "action": "identity",
        "identity": getIdentity()
    }));
};

function showChoosenTask(taskName) {
    const holder = document.getElementById('selected-task-name');
    holder.innerHTML = taskName;
}

function processActiveTaskFunctionBad(task) {
    const result = {};
    let counter = 0;
    let size = input.length;
    return async function () {
        while (counter < size) {
            if (counter < size) {
                let str = input[counter];
                result[str.data] = func(str.data);
                counter++;
                await sleep((Math.floor(Math.random() * slowRatio) + slowRatio
) * 10);
            }
        }
        return {
            action: 'result',
            result: {result: result, task: task.id}
        };
    };
}

function processActiveTaskFunctionDecoratedWithPerf(task) {
    const before = performance.now();
    const result = processActiveTaskFunction(task).call();
    const after = performance.now();
    result.metadata = {
        time: (after - before) / 1000,
        identity: getIdentity(),
    };
    return result;
}

function processActiveTaskFunction(task) {
```

```
    const result = {};
    let counter = 0;
    let size = input.length;
    const sleep = ms => new Promise(resolve => setTimeout(resolve, ms));
    return async function () {
        const before = performance.now();
        while (counter < size) {
            if (counter < size) {
                let str = input[counter];
                result[str.data] = func(str.data);
                counter++;
                await sleep((Math.floor(Math.random() * slowRatio) + slowRatio
) * 10);
            }
        }
        const after = performance.now();
        let response = {
            action: 'result',
            identity: getIdentity(),
            time: (after - before) / 1000,
            metrics: metrics,
            result: {result: result, task: task.id}
        };
        webSocket.send(JSON.stringify(response));
    };
}

function getIdentity() {
    const strings = document.cookie.split(";");
    if (!strings) {
        return null;
    }
    const identities = strings.filter(s => s.trim().startsWith("identity"));
    if (identities.length === 0) {
        return null;
    }
    return identities[0].split("=")[1];
}
```